Enhancing Intervisibility Analyses Using Multi-Computing Techniques

Steve Dowers and Mike Mineter

Institute of Geography,
School of GeoSciences,
The University of Edinburgh,
Drummond Street,
EDINBURGH,
EH8 9XP,
Scotland.

CONTRACT NUMBER N62558-02-M-5605

Final Report

March 2002 – January 2004

The Research reported in this document has been made possible through the support and sponsorship of the U.S. Government through its European Research Office of the U.S. Army.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 2004	Final Mar 2002 to Jan 2004

4. TITLE AND SUBTITLE
Enhancing Intervisibility Analyses Using Multi-Computing Techniques

5. FUNDING NUMBERS
Contract No: N62558-02-M-5605

6. AUTHOR(S)
Mike Mineter and Steve Dowers

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Institute of Geography
School of GeoSciences
The University of Edinburgh
Drummond Street
EDINBURGH EH8 9XP

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

European Research Office
US Army Engineer Research
and Development Center
London, England

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
The report describes research intended to enhance the timeliness and scope for analyses to improve decisions related to intervisibility. The research entailed feasibility studies followed by the design, development and use of prototype software to:
1. Manage the completion of a large number of visibility analyses, each of which determines the visible regions from a point in a Digital Elevation Model. The project accomplishes the visibility analyses by efficient use of spare CPU cycles on a network of processors running Windows.
2. Build a database from these analyses for a test grid of 1201 rows, each with 1201 columns
3. Extract information from the database using a series of tactical decision aids (TDAs) and metrics.
4. Extend the ArcMap application in the ArcGIS product suite, to provide access to these metrics and TDAs.

14. SUBJECT TERMS
terrain, analysis, Digital Elevation Model, DEM, intervisibility, complete intervisibility database, multi-computing, viewshed

15. NUMBER OF PAGES
49

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

CONTENTS

TABLES

FIGURES

0 Executive Summary

0.1 Introduction

This report details the results of research and development undertaken at the Institute of Geography, School of GeoSciences, The University of Edinburgh under contract number N62558-02-M-5605 over the period March 2002 to January 2004. This research follows on from work undertaken during an earlier project (N68171 00 M 5807) referred to here as Phase 1. The aim of the research was to enhance computer-based analysis of terrain to aid decisions relating to intervisibility. This has been achieved by further development of a storage format to allow a complete intervisibility database (CID) to be stored and managed on readily available computers using commonly available tools, together with techniques to allow generation of the CID in a reasonable time frame and tools to perform extraction from and calculations on the CID.

0.2 Enhanced CID storage format

The zip storage format reduces the size of the database by over two orders of magnitude and makes long-term storage of the database practical. Previously researchers had discounted storage of a CID due to storage requirements, preferring instead to perform calculations on the database on the fly and storing the resulting metrics. The CID format also allows various metrics to be stored with the database and the set of metrics may be expanded in the future.

The derivation of a CID starts from a Digital Elevation Model (DEM), which records the height of the terrain at each post, where the posts are arranged in a regular grid. The visibility of each post in a DEM from a particular post can be represented by a Masked Area Plot (MAP), where each post is assigned the value 1 if it is visible and 0 if it is not visible. Figure 1b shows an example of a MAP. The CID is the collection of the MAPs for each post in the DEM.

0.3 Multicomputing architecture for Windows

The calculation of a CID involved a large number of visibility calculations that can benefit from techniques to spread these calculations over a large number of computers. The framework developed in Phase 1 was designed to use groups of computers running UNIX with ArcInfo installed to achieve this. This project extended this capability to computers running Windows NT, 2000 or XP by taking advantage of the Condor software from the University of Wisconsin. This allows the calculation of the CID using ArcInfo to be spread over a large number of PCs utilizing machines when they are not being used by their normal users.

0.4 Tactical Decision Aids

Several tactical decision aids (TDAs) were developed during Phase 1. These allowed extraction and display of MAPs from a CID and calculation of a small number of metrics over the whole CID. These metrics generally consists of a summary statistic for each post in the DEM, such as the number of posts which can be seen. Additional TDAs have been developed during this project resulting in some additional metrics, which can be stored in the extended database format together with the original metrics. The TDAs have been developed as a set of Java classes, which can be reused and extended to develop new TDAs.

0.5 ArcMap extensions

Phase 1 developed a rudimentary Java application to allow the underlying DEM to be displayed and providing interactive access to individual MAPs within the CID. This project has developed extensions to allow access to the DEM, CID and associated metrics from ArcMap, an application within ArcGIS produced by ESRI. This approach allows the CID to be displayed and analyzed in conjunction with any other spatially referenced data, which covers the same part of the earth as the DEM.

The extensions have been developed in Visual Basic (VB) using ArcObjects, which is the underlying technology used within ArcMap. Some of the extensions are concerned with accessing the CID and metrics from ArcMap allowing the display and manipulation within a familiar environment. Data is generally exchanged via temporary files and the display can be saved in a normal map description document. Other extensions are concerned with invoking the TDAs from a Graphical User Interface and allow the resulting metrics to be stored in the CID and then displayed.

The final extensions are intended as exemplars of the type of tools which may be developed through the combination of ArcMap, CID and TDAs. The *target visible* tool allows the user to draw or select lines or areas that define a target. The target is converted to a grid with the same resolution and size as the DEM and the extension then uses a Java TDA to calculate the number of posts within the target can be seen from each post in the DEM. The route animation tool allows the user to draw or select a linear graphic and the application selects a series of points at user-specified equal intervals along the line. The MAPs corresponding to each of these points are retrieved and overlaid on the display. The resulting screen for each point is saved as JPEG image file, which can be animated to demonstrate how visibility changes along the route.

0.6 Some performance figures

The multicomputing architecture was tested with the 336 by 466 cell dataset (Southwest Harbor, Maine) used in Phase 1 (dataset 1) and with a DTED Level 1 dataset of 1201 by 1201 cells of Maine covering the USGS Bangor 1:250,000 quadrangle (dataset 2). A set of 33 PCs, a mixture of 600 and 1000MHz Pentium 4 machines, was used for the generation of the CID for dataset 1 taking 5 hours 48 minutes. A total of 2.47 CPU days was used over this time. A larger set of 39 machines, the additional machines being 1000MHz Pentium 4 machines, was used to generate the CID for dataset 2 in 9.94 days. 330 CPU days were utilized over that time with some of the machines being used by interactive users during the daytime.

The CID for dataset 1 occupied 76.7 Mbytes, a compression ratio of 39 from the raw size of 2.9Gb. A higher compression ratio of 130 was achieved for the larger dataset 2 with the 242Gb of the raw CID being reduced to 1.86Gb.

The time taken to generate metrics using the TDAs is largely a reflection of the large volume of raw data being processed. Even though the compressed dataset is relatively modest, the TDA still has to process 242Gb of data for dataset 2. The cumulative visibility, a simple metric, takes 0.9 hours to generate, while the fragment statistics take 27 hours and the logical metrics take 6 days on a single 2.8GHz Pentium 4 processor.

0.7 Dissemination

A paper and a complementary poster were presented at the 2002 ESRI International User Conference describing the work undertaken during Phase 1. A description of some of the work undertaken during this project was presented through a paper at GeoComputation 2003, supported by a second paper describing analysis and interpretation of the metrics derived from a CID.

0.8 Future directions

Some TDAs, such as *target visible* would benefit from storage of an inverse CID, a database of which posts the observer can be seen from rather than the post which they can see. This can be generated by inverting the matrix held in the CID, possibly distributed over a set of processors using Condor. The processing involved in most of the TDAs is amenable to parallelization by distributing the processing of a block of MAPs to a processor. The Condor environment provides support for Java applications allowing a clear route for this development.

Both CID generation and TDA processing could be developed to take advantage of the growing computing resources through GRID middleware. The process would require some packaging of the existing components to allow invocation as GRID or web services and extension of the multicomputing architecture to utilize GRID resources.

Abstract

The report describes research intended to enhance the timeliness and scope for analyses to improve decisions related to intervisibility. The research entailed feasibility studies followed by the design, development and use of prototype software to:

1. Manage the completion of a large number of visibility analyses, each of which determines the visible regions from a point in a digital elevation model. The project accomplishes the visibility analyses by efficient use of spare CPU cycles on a network of processors running Windows.
2. Build a database from these analyses for a test grid of 1201 rows, each with 1201 columns
3. Extract information from the database using a series of tactical decision aids (TDAs) and metrics.
4. Extend the ArcMap application in the ArcGIS product suite, to provide access to these metrics and TDAs.

1 Overview of body of the report

This document reports on a project undertaken by the Institute of Geography in the School of GeoSciences of the University of Edinburgh, to enhance the computer-based analysis of terrain to aid in decisions related to intervisibility.

The goal of the project was to develop and explore a capability for intervisibility that exploited multicomputing for processors running ArcInfo on Windows, to construct a database of intervisibility data, to develop several demonstration TDAs and multiple metrics for visibility, and to interface these to the ArcMap application within the ESRI ArcGIS product suite.

Following a restatement of the project specification, the report describes each of the major components of the project in turn: the multicomputing architecture, the building of the database, the metrics and TDAs, and the ArcMap interface.

This project builds upon an earlier project, undertaken by the same personnel, under R&D 8707-EN-01 Contract N68171 00 M 5807. To reduce the need for readers to cross-refer to that report, some of its contents are included, with the source stated as the Phase 1 project.

An associated DVD includes this Final Report, all software (with documentation of the Java TDAs), and a complete intervisibility database for the test datasets.

References are made to two datasets: the first was also used in Phase 1, and is 336 by 466 cells; the second is 1201 by 1201 cells. Unless otherwise stated, run times were elapsed times on an otherwise lightly loaded PC, Pentium 4, 2.8GHz, 1GB RAM, running XP with Service Pack 1 installed.

2 Project specification

2.1 Background

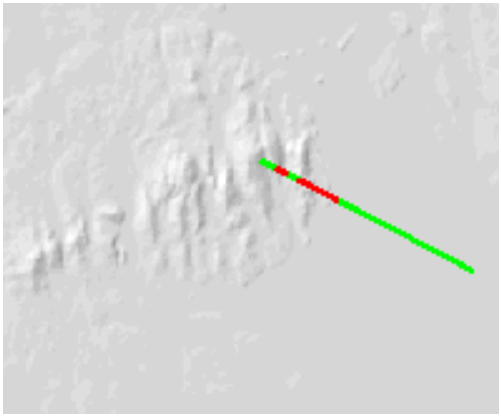
Many geographical applications, including the analysis of terrain, are heavily constrained by processing speeds. The proliferation of large datasets, the potential of Internet/Intranet services, and the use of more sophisticated analyses by integrating multiple datasets all contribute to the present and anticipated bottlenecks in applications across the environmental management, government, commerce and defense sectors. One solution to this constraint involves parallel processing, the use of multiple processors to share in the execution of one task. During the last 13 years the (newly renamed) Institute of Geography (IoG) has followed a strategy that creates frameworks of parallel software to encapsulates the complexity of parallel geocomputation, allowing high-performance applications to be built upon these (Mineter and Dowers, 1999, Mineter, Dowers and Gittings 2000). As the first steps in this strategy, IoG has designed a

number of parallel algorithms to explore the fundamental problems inherent in parallel geocomputation (Healey et al., 1998).

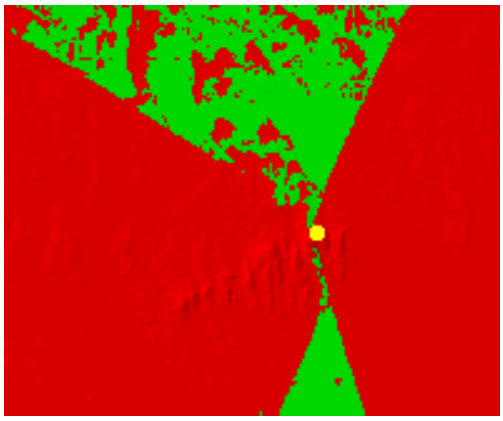
A higher level of parallelism, multi-computing, in which multiple independent tasks run concurrently, one on each of many processors was explored in the 1999-2000 collaboration between IoG and TEC, reported in Mineter et al (2001). Intervisibility analysis is an outstanding example of an application that benefits from multi-computing, as, in the analyses under consideration, visibility maps need to be derived for each point in a digital terrain. Each point can be processed independently, and N points can be processed concurrently if N processors are available. Concurrent processing incurs smaller overheads in inter-processor communication than is the case for low-level algorithmic parallelism.

The background to this project was set by the Phase 1 project. The motivation remains as follows: intervisibility is the term used to describe the effects of terrain on visibility. It is a key factor in military terrain analysis and impacts a soldier's field-of-view, viewing distance, and engagement ranges. A number of different intervisibility products are available. Point-to-point intervisibility results in a line-of-sight profile (Figure 1a). Single point to multiple point intervisibility produces a masked area plot (MAP) (Figure 1b). In both figures, green indicates visibility and red indicates no visibility. Intervisibility products have been largely limited to line-of-sight profiles and MAPs because of the amount of time required to generate individual products.

Figure 1: Line of sight profile and masked area plot (MAP)



Line of Sight Profile (1a)



Masked Area Plot (1b)

The achievements of the first project demonstrated the feasibility of computing and accessing a "Complete Intervisibility Database" (CID): for each post of a Digital Elevation Model (DEM), for a specified observer height, are derived and held the visible regions of the DEM. Issues of computation time, data volume and methods for building applications to exploit the CID arise and were shown to be tractable in the first project, the results of which were as follows for a DEM 336 by 466 cells:

1. Multiple UNIX processors were coordinated to concurrently run multiple visibility analyses (ArcInfo being run independently on each processor) so reduce elapsed run-times for deriving the CID.
2. A structure for the CID was designed. It required 3% of the uncompressed data volume for the trial dataset
3. A Complete Intervisibility Database for a 1:24,000-scale U.S. Geological Survey DEM was generated. The complete database was built in 43 hrs 26 mins using 13 processors, of which 5 were 300MHz UltraSPARCs, the remainder being older Digital Alpha processors. Each processor used in excess of 90% of its CPU time in running the tasks.

4. Three new TDAs were developed, based on the Complete Intervisibility Database

2.2 Goals

1. Establish a capability for deriving a CID using multiprocessing across processors running ArcInfo on the Windows operating system.
2. Extend/modify the CID database to support analyses from a DEM of 1201 columns by 1201 rows.
3. Prototype new applications that access the CID:
    a. Modify existing TDAs
    b. Develop a range of metrics that characterize the visibility surface derived for an observer at each post.
4. Interface the applications to the ArcMap application within ArcGIS.

2.3 Tasks

2.3.1 Task #1: Windows-based Multicomputing Architecture for the CID

The first task was to develop the multicomputing architecture for deriving the Complete Intervisibility Database. The proposed configuration would utilize multiple Windows computers running ArcInfo on a network. ArcInfo would be used for the visibility analysis and data manipulation. This task inherits the requirements of the first-phase UNIX-based system and provides for the following Production Management capabilities

a. Register available computers with appropriate licenses on the network
b. Manage the division of the tasks
c. Determine computer availability for processing
d. Allocate of tasks to multiple computers
e. Monitor the status of on-going jobs
f. Verify that jobs have been completed successfully
g. Report on status of work, i.e., jobs completed, jobs in progress, remaining jobs
h. Incorporate handling of NODATA values, increasing scheduling flexibility, and additional validation checks.
i. Deliver and install software at TEC so that TEC can generate CIDs and demonstrate the application to others.

2.3.2 Task #2: Data Structure and Format for Storing Complete Intervisibility Database

The Complete Intervisibility Database for the phase 2 trial dataset of 1201 × 1201 posts comprises 1,442,401 MAPs, each of 1,442,401 bits, ~2.4×10^{12} bits in total. The methods of compression, storage and access used in Phase 1 (one sub-directory with multiple zip files per row) thus needed to be reassessed and modified extensively.

2.3.3 Task #3: Complete Intervisibility Database

In order to demonstrate the working multi-computer architecture, the University of Edinburgh was to create a Complete Intervisibility Database for a Digital Terrain Elevation Data (DTED) Level 1 DEM, with 1201 rows and 1201 columns. The resulting Complete Intervisibility Database would be generated using the

multicomputing architecture specified in Task #1 and the results will be stored in the data structure and format specified in Task #2.

2.3.4 Task #4: Complete Intervisibility Database Tactical Decision Aids

The Complete Intervisibility Database would be used with exploitation software to generate:

a. a series of Metrics, characterizing the MAPs associated with each post, held as a raster data file:

- Cumulative Visibility (Observed). For each post in the DEM, the cumulative visibility would be calculated. This would be determined by counting the number of MAPs in which the post is visible. (This is a Phase1 existing product)
- Cumulative Visibility (Observer). This would be determined by counting the number of posts visible.
- Core Area (Contiguous area around observer location)
- Fragmentation or Group Count (Number of groups of contiguous areas making up MAP) – where connectivity is defined by either 4 or 8 neighbors.
- Observer/Neighbor Total Area (Area covered by observer and all neighbors)
- A further subtask was to explore alternative approaches to represent Shape and Orientation of the MAP.

b. Post of Maximum Visibility. For each post in the DEM, the post of maximum visibility would be calculated. The resulting table would store the location of the MAP or plots that covers the post and has the maximum number of visible posts. (Also a 1st Phase TDA)

c. Target Visible. For a user-specified polygon, a multi-observed masked area plot would be calculated. This would be determined by summing the MAPs for all locations for multiple observation points. Posts with high values will be locations that are most visible. (Also a 1st Phase TDA)

d. Route MAP Animation. Create the frames for an animation of the MAPs along a route, where a grid similar to the grid used in the Percent Target Visible TDA defines the route. This task will require the creation of layers for each MAP along the route. The user will define a map background in ArcMap and a transparency level for the MAPs. The animation routine will step through the MAPs along the route and perform a screen capture at each point. The screen captures will be numbered <filename>0001.jpg, <filename>0002.jpg, and <filename>XXXX.jpg, so they can be automatically loaded in an animation program.

2.3.5 Task #5: Interface TDA's and CID to ArcGIS.

Create an ArcGIS extension to access the CID and TDAs directly from ArcMap. The following capabilities should be included:

a. Display Masked Area Plot. Using the row, column pair; the x, y coordinate pair, or the cursor, extract a masked area plot from the CID and create a layer in the Table of Contents displaying the masked area plot in red with 50% transparency

b. Incorporate TDAs. Existing and new TDAs, and Point of Maximum Visibility Table should be added to the ArcGIS interface. An interface should be provided to run the algorithms, as well as display the results.

c. The results of the Cumulative Visibility Analysis – Observer, Percent Target Visible, Masked Area Plot Cumulative Visibility, Visibility Metrics, Route Map Animation and should be added as layers in ArcGIS, while the Point of Maximum Visibility should be added as a table.

3 Project achievements

1) The multicomputing architecture has been developed and tested across 37 Windows processors.

   a) Feasibility studies considered alternative approaches:

      i) An initial trial using Cygwin and the existing Phase 1 'task farm' approach failed to provide adequate security. Cygwin provides UNIX-like capabilities on Windows machines, and so would have enabled much of the existing GANNET functionality and code to be reused: in this approach, each workstation requests a new task when it is ready to process it. One processor also acts as coordinator of the task farm. The coordination is achieved using PERL, remote shells, and shared directories. The tasks are performed using local disk space, results being copied to a final directory on a shared disk. However in the feasibility studies of methods for invoking remote shells it was found that the security of the host machines could be compromised due to issues associated with user management, discussed below.

      ii) Explorations of two commercial products (from Entropia and from United Devices) led to recognition of this approach as being attractive but over-expensive for the current budget.

      iii) In late 2002 Condor was released with enhanced support for Windows, and offering many of the features required for remote job invocation and monitoring.

   b) Condor was chosen and a MultiComputing Architecture (MCA) layer built in Perl above Condor, to manage CID generation.

      The resulting system includes the ability to:

      i) Manage which processors are active in any run.

      ii) Distribute visibility analyses to each processor.

      iii) Respond to computer availability.

      iv) Monitor the status of on-going jobs. A combination of Condor and MCA functionality:

         (1) Lists jobs in progress.

         (2) Restarts jobs that have failed (identified by the absence of results, in cases where Condor was unable to identify the failure.)

         (3) Generates statistics concerning use of CPU time

      v) Write statistics to disk periodically, to profile CPU use and task completion across the architecture.

      vi) Re-attach a failed processor to the task farm.

      vii) Close the task farm at a specified time, holding the status of completed jobs for the next occasion when the task farm is used. (Condor allows scheduling of the task farm if desired, for example from 7pm to 7am daily, and over weekends. An alternative mode of use is offered by the running of tasks at low priority.)

      viii) Restart the task farm at a specified time, filling gaps in any missed or previously failed tasks, and then continuing from the run reached previously. (When the task farm is restarted, any tasks that failed are recognized and resubmitted from the PERL scripts – the PERL scripts maintain a list of active tasks, when tasks for a row of the database are no longer active, its results are checked and failed tasks resubmitted). The "specified time" can be achieved by scheduling a batch command.

2) Data structure and Format for storing the Complete Intervisibility Database

This has been designed, implemented and tested. It is described below. It is a development of the Phase 1 design.

3) Complete Intervisibility Database (CID)

Generated for both the first-phase DEM and the 1201x1201 dataset

4) Metrics have been developed and existing TDAs modified

5) The ArcMap application has been extended to interface with the Java TDA and metrics.

6) Four papers and posters were presented at two conferences:

   a) A Multicomputing Software Environment for ArcInfo Intervisiblity Analysis, Mineter, M.J., Dowers, S., Gittings, B.M., and Caldwell, D.R., 2002. In *Proceedings of the 22nd Annual ESRI Conference*. (Redlands, ESRI)

   b) Explorations in Visibility Analysis: Applying ArcInfo in a Distributed Computing Environment, Caldwell, D.R., Mineter, M.J., Dowers, S., and Gittings, B.M., 2002. Poster 22nd Annual ESRI Conference.

   c) Analysis and Visualisation of Visibility Surfaces, Caldwell, D.R., Mineter, M.J., Dowers, S., and Gittings, B.M., 2003. In *Proceedings of the 7th International Conference on GeoComputation*. (Southampton)

   d) High Throughput Computing to Enhance Intervisibility Analysis, Mineter, M.J., Dowers, S., Caldwell, D.R., and Gittings, B.M., 2003. In *Proceedings of the 7th International Conference on GeoComputation*. (Southampton)

   The first two of these reported in more detail on the work undertaken during Phase 1, while the latter pair reported on outcomes from the work undertaken in this project.

4 The MultiComputing Architecture (MCA)

4.1 Overview

The role of the MCA is to execute the many visibility analyses and to use multiple processors to reduce the elapsed time. Each analysis executes on one processor, an analysis being assigned to its processor by a coordinating task. Following a restatement of the Phase 1 MCA, the developments in this Phase 2 are presented, the major goal of Phase 2 being to use multiple processors each running Windows.

During the first phase, an initial demonstrator MCA, subsequently named GANNET (Geographical ANalyses on NETworked computers) was developed to execute many ArcInfo tasks on multiple UNIX processors (Mineter et al., 2002). As shown in Figure 2, a 'task farm' approach was taken: one processor acted as coordinator of the task farm, and each of the remaining processors, termed "workers" executed one task at a time. The worker requested a new task when it was ready to process it, so ensuring the necessary flexibility in response to changing loads on different processors. The coordination was achieved using Perl, remote shells, and shared directories. The tasks were run at low priority to use spare processing capacity, and held all data on local disk space and then copied results to a final directory on a shared disk. A single task comprised the visibility analysis for 16 posts in a DEM, invoked in ArcInfo (installed on each processor) by use of AML scripts. These 16 MAPs were held in one Band Interleaved by Line (BIL) file, termed a MAP16. For analysis of the first trial DEM of size 336 columns by 466 rows, a single task had a typical run time of between 2 and 5 minutes, depending on which processor was used: of the 13 available, 5 were 300MHz UltraSPARCs, the remainder were older, slower processors. Over holiday periods, these were used with efficiencies between 93% and 98% so reducing the elapsed time to 43 hours 26 minutes for the total of 9828 tasks.

GANNET was recoded in this second phase, building on the Condor (2002) middleware (Figure 3). The primary reason for this development was to allow use of processors running Windows. For testing, 37 of these were available in a computing laboratory, and in general are idle overnight and have periods of light use during the day. Recent releases of Condor have enhanced its support for Windows, offering solutions to

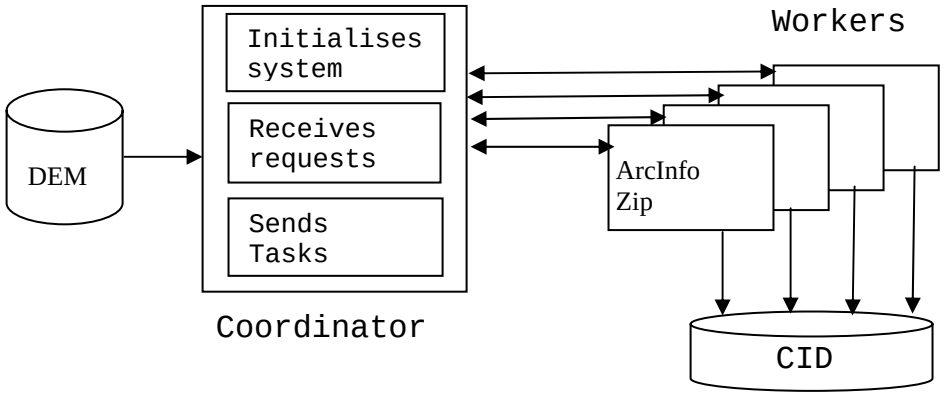


Figure 2: Coordinated use of multiple UNIX processors - Phase 1

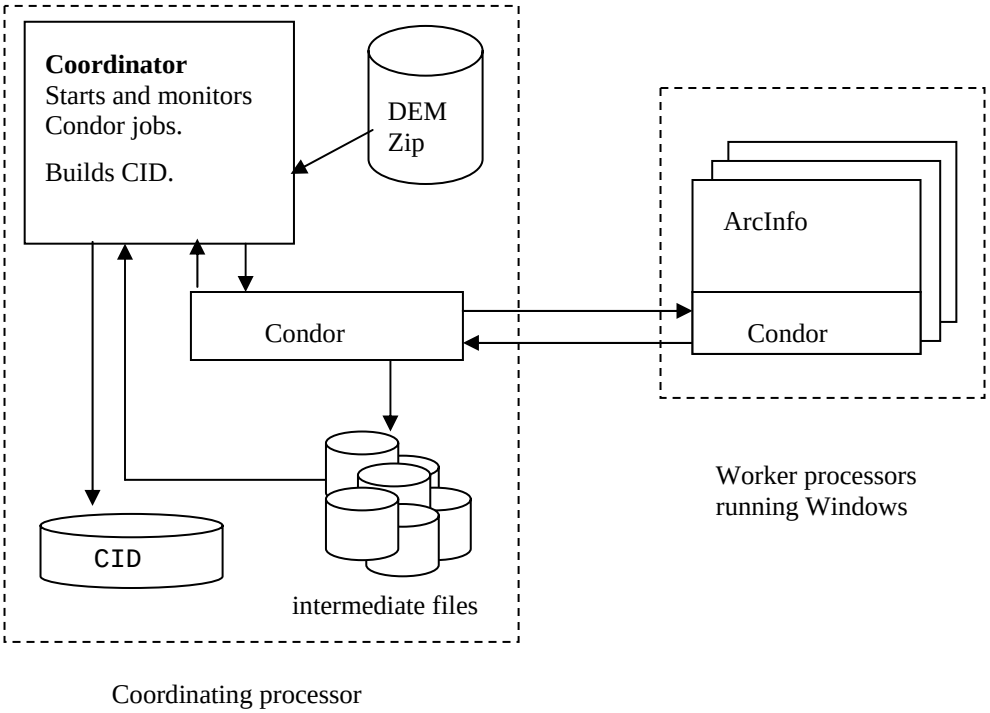


Figure 3: Phase 2 - using Condor with Windows processors

the issues concerning the invocation and management of processes on Windows, and file transfer to and from worker processes. Using Condor also gives longer-term strategic possibilities of a) wide-area distributed processing, with Condor itself integrated with Grid middleware allowing sharing of resources beyond firewalls, and b) of parallel processing, in which case worker processes would in general be required to inter-communicate. In high-throughput computing, as in the analyses described in this paper, the middleware (Condor, here) manages all data exchange: each worker communicates neither with each other nor with the coordinator. Jobs are submitted to Condor, and these are matched to processors according to the characteristics chosen for the processor and the job – so, for example, Condor can be constrained to allocate a job to a processor only when that processor is otherwise lightly loaded; if an interactive user logs on, the Condor job can be terminated. On UNIX, but not Windows, Condor can checkpoint processes and migrate them to less loaded processors.

In the first phase system, the overhead of starting a new task was negligible in comparison to run times, even when a task comprised only 16 visibility analyses (for observers in a 4x4 sub-grid) creating one MAP16 BIL file. In the second phase with Condor, the overhead was reduced by configuring Condor jobs to comprise multiple tasks, and Condor itself seemed more stable with a limited length of job queue. For an initial run with the trial dataset, the USGS DEM from the Phase 1 effort (336 by 466 cells), 21 tasks were executed per Condor job so that 4 Condor jobs execute the analyses associated with each row of sub-grids. The full CID analysis for the Phase 1 trial dataset was accomplished in about 3 hours 26 minutes: times will vary according to the availability of the 37 processors. 40 jobs were maintained in the Condor queue, so that free processors were allocated a new job by Condor without application-level code being executed.

4.2 Condor

4.2.1 Overview

Condor is freeware developed by the University of Wisconsin over more than a decade. It is used in this project to provide a means for invoking tasks on PCs, when those PCs are otherwise lightly loaded. It is the basis on which the MCA is built.

A Condor coordinator task, running on a specified machine, receives requests from any suitably configured PC. These requests specify tasks that need to be run. When a resource (a PC) is available, the coordinator invokes that task on the PC. Thus Condor insulates the CID specific and MCA functions from the management of the machines: Condor recognizes when a machine is running, busy, or available.

Tasks are assigned by Condor from a coordinator process; each PC has a Condor process always running to communicate with the Condor coordinator. When specified requirements match a machine's state, then the coordinator causes the PC's Condor task to initiate a new task; it receives data and executables, invokes the required tasks and sends results to the machine that submitted the request to Condor.

4.2.2 Installation

Initial installation of Condor was carried out using the distribution for Windows NT packed as a single executable installation program. This was carried out on the central manager machine PC first and then one other PC which would be a member of the pool. The installation procedure, including system requirements, is described in section 6.3 of the Condor manual. Section 6.3.3 describes installation using the setup program and the responses required during the installation process. The most important of these for the central manager PC (showing example responses from the configuration in Edinburgh) are:

- Name of the condor pool: UEdin-Geog-1
- Hostname of this machine: geod164.geo.ed.ac.uk
- Size of pool: More than one machine
- Roles for this machine: Allow machine to submit jobs and execute jobs

- Where will condor be installed: c:\Condor
- Where should Condor send e-mail if things go wrong? Steve.Dowers@ed.ac.uk
- The domain: geo.ed.ac.uk
- Access permissions:
    - Read: *.geo.ed.ac.uk
    - Write: *.geo.ed.ac.uk
    - Administrator: geod164.geo.ed.ac.uk, geop20.geo.ed.ac.uk
- Job start policy: After 15 minutes of no console activity and low CPU activity
- Job vacate policy: The job is killed 5 minutes after your return

These responses are used to set values in the condor_config file, which can be changed and adjusted after configuration. The installation for a member of the pool is similar but does not require the name of the pool or the size of the pool.

Section 3 of the Condor manual describes the configuration options which may be changed in the condor_config file. Initial values for these entries are set by the installation program but alternative values were set after initial testing of the condor pool with visibility generation using ArcInfo. The values that were changed relate to the start policy and the vacate/kill policy to make effective use of the resources available but without interfering with normal interactive use of the PCs in the pool. These steps were necessary due to the way in which ArcInfo starts processes, described below under ArcInfo issues. The parameters were also adjusted to give a shorter time between the interactive user leaving the machine and the Condor job being started. The parameters changed were:

- StartIdleTime = 5 * $(MINUTE)
- WANT_SUSPEND = TRUE
- WANT_VACATE = FALSE
- START = $(UWCS_START)
- SUSPEND = FALSE
- CONTINUE = $(UWCS_CONTINUE)
- PREEMPT = FALSE
- MaxVanillaSuspendTime = 60
- MaxVanillaVacateTime = 70
- SUSPEND_VANILLA = ($(KeyboardBusy) || \
  ((CpuBusyTime > 2 * $(MINUTE)) && $(ActivationTimer) > 2))
- CONTINUE_VANILLA = ($(CPUIdle) && ($(ActivityTimer) > 10) \
  && (KeyboardIdle > $(ContinueIdleTime)))
- PREEMPT_VANILLA = (((Activity == "Suspended") && \
  ($(ActivityTimer) > $(MaxVanillaSuspendTime))) \
  || (SUSPEND_VANILLA && (WANT_SUSPEND == False)))
- KILL_VANILLA = $(ActivityTimer) > $(MaxVanillaVacateTime)

Section 6.3.4 of the Condor manual describes the procedure for carrying out a manual installation of Condor on Windows NT, basically consisting of copying the files for the C:\Condor directory and running the supplied install program to install and configure the condor_master service. The one PC installed from the setup program was used as the master for this process and the files and installation transferred to the other machines in the pool using a batch job submitted to the set of machines using normal management tools.

Although the Condor manual implies that Condor has not been tested under Windows 2000 and Windows XP, installation was carried out successfully under both these operating systems and during CID generation, the submitting machine, (and therefore the machine running the shadow processes to deliver and collect files from the pool) was running Windows XP while the remainder of the pool were running Windows NT.

4.2.3 Licensing issues

A Condor pool can contain processors with various operating systems. For CID purposes, it is also necessary that processors used have ArcInfo licenses. The application_set.sub file, used as the basis for a Condor submit file, can include selection of the processors operating system. It cannot specify that an ArcInfo license be available. The assumption is currently made that processors matching the submission requirement do have ArcInfo licenses. If this is not he case then jobs will fail rapidly, and many jobs will repeatedly be assigned to the unlicensed processor.

Condor pools can receive jobs from each other (termed "flocking"). Therefore processors with ArcInfo licenses could be in one pool. Alternatively, the "class ad" used to advertise a processor to allow matching with submitted tasks, could have an additional attribute defined, this being used in the MCA file application_set.sub, described below.

If the licenses for ArcInfo are managed through a license server providing floating licenses, a static class ad mechanism would not be adequate and a more sophisticated daemon to monitor license usage would be required. This could be implemented either at the level of individual pool members or at the level of the coordinator. By implementing queries of the central license server, the coordinator could monitor the number of licenses available and could manage jobs in the Condor queues to maintain a specified minimum level ArcInfo licenses to be available to interactive users.

4.2.4 Issues with ArcInfo

The way in which ArcInfo starts processes to execute commands presented some problems when running them in the Condor context. Instead of passing on the low CPU priority which is correctly set by Condor, ArcInfo starts its sub-processes at normal user priority which has a significant effect on the response for an interactive user. Investigations into options to alter the priority of the sub-process under the NT platform were not successful, so the Condor configuration parameters were altered to kill the job as soon as the interactive user started to use the PC again.

4.2.5 Condor commands used

condor_submit: from CondorMCA.pm, to queue a new cluster of jobs.

condor_q: from CondorMCA.pm, to monitor queued and running jobs

condor_status: Not called from Perl, used occasionally to monitor all processors in the Condor pool

condor_rm: To remove an apparently hung job. Used from the command line, not run automatically by MCA. This was needed when Condor failed to terminate a suspended process, following logon to a machine of an interactive user. When a Condor job is halted by condor_rm, the MCA recognizes the job has stopped, tests for its results and resubmits for any missing data.

4.3 Design and Implementation

4.3.1 Requirements and concepts

The MultiComputing Architecture (MCA) for Windows coordinates the queuing and execution under Condor of many tasks. As stated above, these differ from each other only in aspects that can be derived from the task identifier (a number in the range from 1 to the total number of tasks).

The concept of a set of tasks was defined: a set is a collection of tasks – in the case of intervisibility, it is the tasks for creating data for a complete row of the CID. A set of tasks is executed in one or more clusters: when all clusters for a set of tasks have ceased to execute then the results of the set are checked and any gaps filled. Gaps are in general associated with jobs that were prematurely terminated, for example due to other activity on the jobs' processors.

The Condor perspective is thus one of jobs in clusters. These are started by submission of a ".sub" file, using the condor_submit command, called from the MCA Perl script. The application perspective is of tasks in sets.

The MCA brings these perspectives together. The design emphases were:

1) To allow Condor to maintain activity on all available processors, whilst avoiding excessively long queues of pending jobs in Condor. This proved problematic a) according to Condor documentation and b) from experience of a coordinating machine hanging, which has not recurred with smaller job queues. By use of the set concept and queuing only a few jobs more than there are processors available.
2) To be resilient to closure and restart of the MCA. By checking on startup for the existence of text files giving the current status, these being written at every event in MCA.
3) To allow MCA closedown/startup to be automated, e.g. from a timed batch script. MCA checks for he existence of a file named "*application*STOP.txt" e.g. CIDMaineSTOP.txt. If this is found then no more jobs are submitted to Condor. Those already submitted are allowed to terminate normally. (A script used to create CIDMaineSTOP.txt could also execute condor_rm to terminate jobs prematurely.)
4) To respond to Condor closing jobs before all data are derived. By checking data generated when all jobs in a set leave the Condor queue.
5) To be runable from a Windows machine. By using Perl, DOS scripts and ArcInfo AML scripts.
6) To allow multiple instances of the MCA to execute concurrently. By encoding the application name in relevant files. It is unlikely that this is desirable, due to competition for resources.
7) To allow other Condor jobs to be executing or queued. Condor can be used for the running of individual processes, not only for high-throughput of multiple similar processes as in MCA. Achieved by knowing which Condor jobs are associated with the MCA.
8) To allow easy monitoring of results as the processes are run. Files for each set are written into their own subdirectories.
9) To allow easy monitoring of status as the processes are run. The status can be found in txt files written by MCA, in trace from MCA and from condor_q and condor_status commands.

4.3.2 Implementation

4.3.2.1 Preparation for CID creation

The zip file *preparation.zip* contains:

1. A directory with scripts and executables used to take two DEMs and build the directory from which the MCA is run.
2. A subdirectory templateRun containing the MCA scripts.
3. A subdirectory CIDDATA used to temporarily hold the ArcInfo grids derived from the DEMs
4. A subdirectory TEMP used for other temporary files.

The script *prepareSA1.bat* invokes the Perl used to set up the MCA. In this case the *SA1* refers to the name of an example CID, and *CIDSA1* is the name used by the MCA to identify all Condor clusters for this CID.

Two DEMs are used (rather than the single DEM in Phase 1):

1. the surface used for the visibility analyses
2. the surface used to give observer elevations.

In the examples run for this project, these are identical; in a demonstration run for a small project for a UK company, the first DEM was modified by adding elevations of features, such as buildings: these obstruct the view, but users are assumed not to stand on the roof; checks are in place so that users "under" buildings have null MAPs. Other features might comprise wall heights, or forest heights.

The Perl script *install.pl* creates a top directory for use by the MCA, and the zip file later used by MCA and Condor to send data and executables to the PCs that run tasks. The arguments for *install.pl* are:

1. cidname: String used to identify CID, must begin with CID to satisfy the MCA constraints.
2. ESRI grid dataset name: observer DEM
3. ESRI grid dataset name: surface DEM (modified by features otherwise the same as the observer DEM)
4. The directory that holds both the DEMs
5. The size of the Condor pool: used to determine how many tasks are queued to Condor.
6. The name of the directory to be used for running the MCA

The Perl script invokes an ArcInfo AML, *installdata.aml*; this copies the grid datasets and renames them; it should create an ArcInfo GRIDASCII format file named *dtm* from the surface DEM, and a GRIDASCII file *obsdtm* from the observer DEM. However the command "arc: gridascii <obs_grid_file> obsdtm" and the corresponding one for *dtm*, failed except when invoked from the script: they should be used by hand from the command line to create the GRIDASCII files in the directory that holds both DEMs (argument 4 above).

4.3.2.2 The MCA and CID creation

The Condor jobs create intermediate files which, when all are present, are post-processed into the final CID. The intermediate files comprise a large number of zip files, held in subdirectories specific to the set. The MCA is run in the "top directory" and data are collated into .\data below that. Individual Condor job output is written to the .\data*set_number* directories and all Condor logs are written to one file, named in the .sub file (below), generally resident in the top directory.

The directory created by the preparation scripts contains files in three sets:

1) generic MCA functionality
2) all CID applications
3) a specific CID application (creation of a CID for a particular DEM)

The generic MCA modules are as follows:

- *MCACondor.pm*: the coordinating MCA-Condor Perl module
    - its main function, CondorAMCA is invoked from Run*cidname*.pl, for the 1201x1201 dataset
    - CondorAMCA arguments:
        - cidname: a concatenation of an application-class and instance name. e.g. CIDMaine was used for the first dataset, CID2 for the second. CID-specific modules are invoked by MCACondor when the application name begins with "CID". (The only alternative application-class available in MCACondor is "Test" for development purposes.)
        - Condor results top directory – usually "." has been used to date, the Perl scripts etc being copied to this directory before MCA is run.
        - Directory for the final CID – currently unused as creating the final CID is decoupled from the running of the Condor jobs.

- *MCACluster.pm*: object that encapsulates cluster-set correlation. An array of these objects is held by MCACondor which periodically checks the list against the list of active clusters/jobs from the condor_q command.

The application class ("CID") modules are:

- *CidMCA.pm* Object oriented module invoked from MCACondor code. Interfaces are:
    - new, invoked to construct the CidMCA object.
    - setUp($application, $postCondor,$postSink, $TasksPerSet,$finalTask) the inputs to the function being:
        - $application: "CIDMaine" or alternative
        - $postCondor: directory of the data returned by Condor
        - $postSink: directory where final data reside (the top directory of the CID, in this case)
        - $tasksPerSet: max. number of tasks in a set
    - sinkProcessing($doneSet) returns an array with ranges of any missing tasks, arranged pairwise, to give the first and last missing task in the set: e.g. {10,12,83,84} denotes two missing ranges, 10-12 and 83-84.
    - Checks for missing tasks are carried out:
        - When all tasks in a set have stopped running, to check the set and (optionally – see below) resubmit missing tasks.
        - For all sets on completion of the final set, resubmitting tasks for missing data.
- *CIDWriteToNew.pl* Creates the CID from the data files returned by the Condor jobs.
    - arguments:
        - topIntermediateDir: usually subdirectory .\data; the directory holding data returned by Condor jobs.
        - finalCIDdir: directory to hold the CID.
        - number of columns in DEM
        - number of rows in DEM
- *cidTest.pl*: scans intermediate files created by Condor jobs to find gaps, multiple log files, multiple .sub files (see below for explanations). Prints all exceptions found. (Does not initiate any new tasks.)
    - arguments:
        - CIDname: CIDMaine, CID2 etc
        - Directory holding the application's control files.
        - topIntermediateDir: usually subdirectory \data
        - finalCIDdir – currently unused
- *cid.aml*: the ArcInfo aml that loops to perform visibility analyses for a sequence of tasks. Invoked by batch file which is specified in the Condor submit file. Input arguments:
    - Name of Arc grid dataset holding the surface DEM.
    - Name of Arc grid dataset holding the observer-height DEM
    - First task in cluster

- Number of tasks per job
- Process number
- Last task in cluster

The visibility command is run from this file: it is here that offsets other than the default 1m for an observer's height would be set. The executable endian.exe is invoked from the AML to ensure that the resulting data are held in big-endian form.

- *zip.exe*, *unzip.exe*: used to unzip the DEM and zip the results, so reducing the network traffic.
- *endian.exe*: Changes the endianism of the *.bil* to big-endian as used by the Java classes that read the CID.
- Notes:
  - CIDWriteToNew could be integrated with CidMCA to do post-multicomputing, final stage sink processing. If that integration is done, then the call to CIDWriteToNew should reside in CidMCA not CondorMCA.pm, as it is a CID-specific function.
  - Use / not \ for directory delimiters in arguments to Perl.

The files specific to a particular CID derivation are:

- DEM grids (as described above)
- *"application"_set.sub* e.g. *CID2_set.sub*. To this *.sub* file MCA adds extra lines to create the Condor submit file that, with the condor_submit command, queues each cluster – the extra lines being the arguments to the *CID2arc.bat* file and the "queue n" line that determines the number of tasks in the cluster. This file names the *.zip* file that contains the DEM grid (so that file is transferred by Condor to the host), and the *CID2arc.bat* file to be executed on the host machine.
- Temporary files used by CondorMCA to monitor Condor jobs. These have encoded within them the application name (CID2, CIDMaine) so that the possibility of multiple MCA runs being concurrent (at the cost of performance to both) is feasible. Files, rather than in-memory structures, are used so that MCA can be restarted after failure – including power cuts, as was accidentally demonstrated in testing…. These temporary files are:
  - *"application"_clusters.txt*, a list of the current clusters queued from CondorMCA. This is refreshed in every iteration of the main CondorMCA loop, currently every 10 seconds. On statup of the MCA, this file is looked for, and read if found, so that the MCA can continue from a previously terminated session.
  - *"application"_condorq.txt* Created by the MCA in each iteration, by calling the condor_q command, to list the jobs in Condor.
  - *"application"_status.txt* Writes the number of the last submitted task. On MCA startup, this file is looked for and if it exists, read to restart the MCA from the next task.
- DOS batch file e.g. CID2arc.bat. Invoked from Condor, named in the ".sub" file above. This invokes ArcInfo and includes the name of the grid holding the DEM. Arguments to the batch file:
  - Process number (determined by Condor)
  - Cluster number (determined by Condor)
  - First task in the cluster
  - Number of tasks in the cluster
  - Set number. Used for trace only.

    - Last task in the cluster. (So that subsets of a cluster can be re-run if needed.)

- *"application"_control.txt*. Specifies the total number of tasks, number of tasks per set, number of tasks per job. Created by install.pl (above); this can be edited to change e.g., allocation of tasks and jobs to processors.
- Run_ "application".bat: causes the MCA to start for the CID.
- *Stop_CID.bat* Causes the MCA to stop submitting new tasks to Condor: does not abort current or queued tasks.
- Header: derived from the GRIDASCII representation of a DEM, gives the number of rows/columns.

The MCA manages the recognition of failure of any tasks, resubmitting tasks if they are not recoverable by Condor. In early tests resubmission was only needed after a Condor job failed to complete, in our cases either because the owner of the processor logged on and was given priority, or because a particular PC was included in the Condor pool although it lacked the necessary ArcInfo license. The first cause of failure was addressed as described below, by a combination of Condor options and additional programming to control ArcInfo priorities. Once the failed PC was removed from the Condor pool, no further failure occurred. Further processing collates the results of the Condor jobs to build the desired datasets, in the present case the CID. Tasks differ from each other only in aspects that can be derived from the task identifier: in the case of intervisibility, the task identifier is used to obtain the location of the 16 observers for which visibility is derived in the task; in the case of an environmental model, it might be a random variation of a parameter that is required.

Tasks are submitted to Condor in clusters: a cluster can comprise 1 or more jobs, and each job comprises one or more tasks. Each job is allocated to a different processor. The number of jobs per cluster is specified in a Condor "submit file", a text file that also specifies the type of processor to be used, the software to be executed, and files to be copied to the target processor before a job is run, and from it to retrieve results. The cluster number is determined by Condor, and is incremented at each submission from the same user. It can subsequently be used to monitor the status of jobs, and to retrieve the history of executed jobs. There is no inter-process communication, so the task farm of the UNIX implementation in GANNET is replaced by one coordinating Perl script that monitors the number of jobs already queued or active in Condor, and submits new clusters to Condor as jobs are completed, so ensuring that all processors can be occupied. Condor manages the queue of requested jobs – there is no direct communication between the coordinating Perl and any worker processor.

Certain constraints apply to current (v. 6.4.7) release on Windows, notably that networked drives cannot be used by Condor. In general, a CID will be required to reside on a shared disk. The data derived by Condor are therefore written into intermediate files in subdirectories on the coordinating machine. On completion of the multiprocessing phase, these intermediate files are zipped and copied to the CID directory, a task requiring a few minutes. The format of the CID is discussed in section 5.

4.3.2.3 Invocation of the Visibility analysis

An ArcInfo AML file is invoked by a DOS batch command, the name of which is passed to Condor when a job is queued

The AML is executed locally on each worker, using local disks:

- Header file: gives the size of the grid; created by install script.
- Input data directory with point and grid data, set up by the same install script.
- The task number passed by the coordinator task farm
- A simple algorithm to derive a bounding box that contains 16 points, and extracts these points from the point-database into a temporary database. It also derives the block row and column.

The processing is thus:

1) From the task number and the file header, derive block bounding box and top-left coordinates of grid block.
2) Select points within the block bounding box from the point database
3) Run the visibility analysis
4) Convert the resulting grid to a BIL image
5) Compress the image using zip
6) Delete temporary files

The above steps are executed for multiple tasks, in a range determined by parameters passed to Condor, and via the DOS batch command to the AML.

4.3.3 MCA configuration

Configuration concerns the determination of sets and the allocation of tasks to Condor jobs and clusters.

The preparation script, install.pl, receives the Condor pool size as a parameter and also estimates an initial configuration held as already described in a text file, "application"_control.txt.

4.4 Execution and Performance

In an early run with the phase 1 (336 by 466 cell) dataset, with 84 tasks per Condor job, and 117 jobs, a total of 3569.17 mins (2.47 CPU days) was harvested in 5 hours 48 minutes. 33 processors were active; these ran between 2 and 6 jobs each. 6 of the jobs failed (on different processors), and required some of the tasks to be resubmitted. The failures were related to Condor installation, leaving jobs suspended when users logged on to their PCs. This problem was fixed during the derivation of the larger CID for the second dataset.

In deriving the CID for the second DEM (1201 x1201) 40 PCs were initially used, one of these being removed due to ArcInfo being unavailable: when a task fails in a manner that Condor cannot diagnose then a new task is assigned to the PC – many such tasks failed and were resubmitted by the MCA. The Condor log reports that 330 CPU days were used on workers. The runs were completed in 9.94 days.

301 tasks ran ArcInfo for results from each of 301 rows of MAP16s; the 301 tasks per row being split into 19 Condor jobs (18 of 16 tasks, one of 13 tasks). Typically 2 or 3 rows were being processed concurrently to occupy up to 40 processors.

Snapshots were taken of several files during the run, named "CID2" and are as follows.

CID2cluster.txt: current tasks in Condor: (each line is Condor cluster, number of jobs, MCA set number, first task, last task)

18894 19 285 85485 85785

18895 19 286 85786 86086

18896 19 287 86087 86387

18897 19 288 86388 86688

The trace written to standard output by the MCA is as follows:

Wed Aug 20 14:55:45 2003

MCA submitted 4 clusters:

MCACluster 18894	njobs: 19	set: 285	tasks: 85485 - 85785
MCACluster 18895	njobs: 19	set: 286	tasks: 85786 - 86086
MCACluster 18896	njobs: 19	set: 287	tasks: 86087 - 86387
MCACluster 18897	njobs: 19	set: 288	tasks: 86388 - 86688

.....

This is updated every 10 seconds, when the Condor queue is checked. If no changes are apparent, then a "." is added to the trace to show continuing sign of life.

The command "condor_q –run" gives currently active jobs:

-- Submitter: GEOD183

ID	OWNER	SUBMITTED	RUN_TIME	HOST(S)
18894.9	mjm	8/20 10:55	0+02:47:29	geod116.geo.ed.ac.uk
18894.14	mjm	8/20 10:55	0+02:29:22	geod126.geo.ed.ac.uk
18894.18	mjm	8/20 10:55	0+02:28:08	geod158.geo.ed.ac.uk
18895.1	mjm	8/20 12:20	0+02:12:55	geod147.geo.ed.ac.uk
18895.10	mjm	8/20 12:20	0+01:38:55	geod118.geo.ed.ac.uk
18895.12	mjm	8/20 12:20	0+01:34:08	geod164.geo.ed.ac.uk
18895.16	mjm	8/20 12:20	0+01:14:23	geod132.geo.ed.ac.uk
18895.17	mjm	8/20 12:20	0+01:14:55	geod142.geo.ed.ac.uk
18895.18	mjm	8/20 12:20	0+01:13:56	geod124.geo.ed.ac.uk
18896.0	mjm	8/20 13:31	0+01:12:27	geod121.geo.ed.ac.uk
18896.1	mjm	8/20 13:31	0+00:59:32	geod111.geo.ed.ac.uk
18896.2	mjm	8/20 13:31	0+00:57:12	geod134.geo.ed.ac.uk
18896.3	mjm	8/20 13:31	0+00:43:49	geod150.geo.ed.ac.uk
18896.4	mjm	8/20 13:31	0+00:21:05	geod133.geo.ed.ac.uk
18896.5	mjm	8/20 13:31	0+00:25:48	geod112.geo.ed.ac.uk
18896.6	mjm	8/20 13:31	0+00:25:54	geod148.geo.ed.ac.uk
18896.7	mjm	8/20 13:31	0+00:21:10	geod152.geo.ed.ac.uk
18896.8	mjm	8/20 13:31	0+00:09:17	geod140.geo.ed.ac.uk
18896.9	mjm	8/20 13:31	0+00:05:31	geod157.geo.ed.ac.uk
18896.10	mjm	8/20 13:31	0+00:00:28	geod138.geo.ed.ac.uk

The "condor_q" command lists all running and idle tasks:

-- Submitter: GEOD183 : <129.215.84.222:1031> : GEOD183

ID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD

18894.9 mjm 8/20 10:55 0+02:48:02 R 0 37.5 CID2arc.bat 9 1889

18894.13 mjm 8/20 10:55 0+02:33:36 R 0 37.4 CID2arc.bat 13 188

18894.14 mjm 8/20 10:55 0+02:29:55 R 0 37.4 CID2arc.bat 14 188

…..

18896.10 mjm 8/20 13:31 0+00:01:01 R 0 0.0 CID2arc.bat 10 188

18896.11 mjm 8/20 13:31 0+00:00:00 I 0 0.0 CID2arc.bat 11 188

…

18897.17 mjm 8/20 14:55 0+00:00:00 I 0 0.0 CID2arc.bat 17 188

18897.18 mjm 8/20 14:55 0+00:00:00 I 0 0.0 CID2arc.bat 18 188

47 jobs; 27 idle, 20 running, 0 held

The command at the end of each line is truncated, incidentally.

To monitor the number of active processes running under Condor, the Perl script moniCondor.pl was written. An example of its output is as follows, with the second column indicating the number of active processes:

Epochtime, nprocesses, date-time

1061384923 30 Wed Aug 20 14:08:43 2003

1061385223 29 Wed Aug 20 14:13:43 2003

1061385523 29 Wed Aug 20 14:18:43 2003

1061385823 30 Wed Aug 20 14:23:43 2003

1061386123 29 Wed Aug 20 14:28:43 2003

1061386423 29 Wed Aug 20 14:33:43 2003

1061386723 28 Wed Aug 20 14:38:43 2003

1061387023 26 Wed Aug 20 14:43:43 2003

1061387323 26 Wed Aug 20 14:48:43 2003

1061387623 26 Wed Aug 20 14:53:43 2003

1061387923 24 Wed Aug 20 14:58:43 2003

1061388223 24 Wed Aug 20 15:03:43 2003

1061388523 21 Wed Aug 20 15:08:43 2003

1061388823 22 Wed Aug 20 15:13:43 2003

1061389123 21 Wed Aug 20 15:18:43 2003

1061389424 20 Wed Aug 20 15:23:44 2003

1061389724 20 Wed Aug 20 15:28:44 2003

1061390024 20 Wed Aug 20 15:33:44 2003

1061390324 23 Wed Aug 20 15:38:44 2003

1061390624 25 Wed Aug 20 15:43:44 2003

5 Building the Complete Intervisibility Database

5.1 Overview

The concepts are unchanged from Phase 1:

- Input is a DEM comprising a regular square grid containing elevations at each a post in the centre of each grid element. There are nCols x nRows posts in the grid. Rows are numbered from the top of the grid in the normal image-processing convention.
- A MAP for the "observer" post at column i and row j, MAP(i,j) contains a binary value for each post in the grid indicating whether that post is visible. The MAP therefore contains nCols x nRows binary values and the simplest representation is an nCols x nRows array of bits.
- A Complete Intervisibility Database (CID) comprises a collection of MAPs, data showing which posts are visible from each post of a DEM. A CID is a collection of MAP(i,j) for each post in the original grid. There are therefore nCols x nRows MAPs in the CID. The total size is nCols x nRows x nCols x nRows bits and so some form of compression is desirable.
- ArcInfo can process up to 16 observer points in one command, the result being stored in an integer grid with 16-bit values where each bit represents an observer. By selecting the 16 observers in a 4x4 subgrid of the original grid we can expect there to be significant correlation between the bits in a 16-bit word, which will aid in compression. The post processing of a MAP can also be simplified with this form of representation since each 16-bit value corresponds to a post and therefore post-by-post processing is simplified compared with unpacking single bit values. The collection of 16 MAPs together is called a MAP16 and is identified by the column and row of the top-left post. There are therefore (nCols/4) x (nRows/4) MAP16s in the CID. For a 336 by 466 input grid there are 9828 MAP16s. The MAP16s are generated in binary form by converting from ArcInfo grid to BIL format.
- The BIL files for the MAP16's are compressed using zip.
- The database structure is required to support simple random access to any MAP16 and expansion is achieved using standard tools available on almost all computing platforms.

In the first phase, the Complete Intervisibility Database was a collection of directories, containing files where each holds 16 MAPs. Each file held a 16-bit integer grid, the k'th bit being part of the k'th MAP. The grid was converted to a Band Interleaved by Line (BIL) format, compressed using zip and copied to the appropriate directory in the CID. The folder and file name was determined from the location of the 16 observers. By selecting the 16 observers in a 4x4 sub-grid of the original grid, there was significant correlation between the bits in a 16-bit word, and so improved compression. This database structure had the advantage of simple random access to any BIL file and expansion could be achieved using standard tools available on almost all computing platforms. In a trial for an input DEM of 336 columns by 466 rows, the compression reduced the dataset size to 86.5Mbytes, ~3% of the uncompressed size (2.9Gb). In this case the CID comprised 9828 files, with 84 files in each of 117 directories (one directory per sub-grid row).

While this structure had the advantage of simplicity, it had the disadvantage of having a large number of relatively small files. The number of files grew with order $O(n^2)$ so this number would increase substantially with large DEMs. On file systems where the allocation unit or cluster size is significant, there may be a substantial amount of wasted space. In addition, there may be limits on the number of files that may be stored on a file system. To avoid this problem, a second version of the database was developed where the BIL files for a complete row are collected into a single zip file rather than (NCols/4) separate files. The names of the entries within the zip file are generated from the column of the top left post in the BIL (0_im.bil, 4_im.bil ...). The file system therefore sees (NRows/4) zip files rather than (NRows/4)*(NCols/4) files and (NRows/4) directories. For the CDROM media where the allocation unit is 2Kbytes, the trial DEM of 336 columns by 466 rows occupies 76.7 Mbytes in version 2 format as opposed to 86.5 Mbytes in version 1 format. The Java code used to read the CID was modified to recognize the version 2 format database in addition to the version 1 format and slightly optimized to minimize repeated

opening of the same row's zip file. This resulted in a slight improvement in the underlying time to read a complete CID.

5.2 Database design

The database comprises one directory in which is held:

1) a zip file for all MAP16 in a row, named *row_nn.zip*, where nn is the first MAP1 row in the MAP16s, so the first two zip files are row_0.zip, row_4.zip. Each zip file contains *.bil* files, named *cc_im.bil*, where cc is the column number of the first MAP1 in the MAP16: the first two being *0_im.bil*, *4_im.bil*.

2) header files describing the original DEM, projection information and a missing value mask.

To these are added:

3) a subdirectory \metrics, holding all metrics

4) a directory to hold layer files for display in ArcMap of data from this CID, \metrics\layers.

These additions are described in section 6 and 7 respectively

5.3 Database construction

The MCA produces intermediate data files that need reorganizing in to the final CID form, achieved by the PERL script CIDWriteToNew. Typical invocation of this final step is from the command line:

CIDWriteToNew.pl data u:\mycid\database 1201 1201

where the script is run in the directory above the intermediate data files (held in ..\data), the CID folder must be a complete filename (not relative), and the last two arguments are ncols and nrows respectively.

The intermediate files reside in one directory tree (.\data in this example) on the coordinating process's filespace (as Condor does not presently support networked drives on Windows); the final CID is not so constrained. The intermediate files are held in subdirectories, one per row of MAP16s.

5.4 Database size

5.4.1 336 columns by 466 rows dataset:

The final database, with each file separately compressed, is 76.7 Mbytes (in version 2 format of the database) i.e., approx 2.5% of the raw 4-D bitmap size (2.9Gb), a compression ratio of 39. It comprises 117 zip files each with 84 entries (one entry per column). Figure 4 shows a shaded relief map of this dataset

5.4.2 1201 by 1201 dataset:

The resulting zip files total 1.86 GB (2,007,048,519 bytes), 0.77% of the uncompressed 1201**4 bits (2,080,520,644,801 bits = 260,065,080,601 bytes), a compression ratio of 129.58. Runtime for converting the Condor output to the final CID was 1004 seconds. A shaded relief map of this dataset is shown in Figure 5.

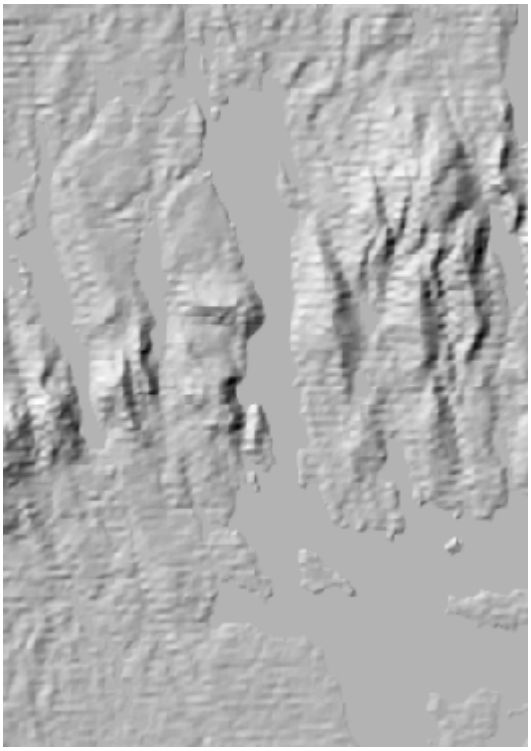


Figure 4: Shaded Relief Image of the Phase 1 data set for Southwest Harbor, Maine

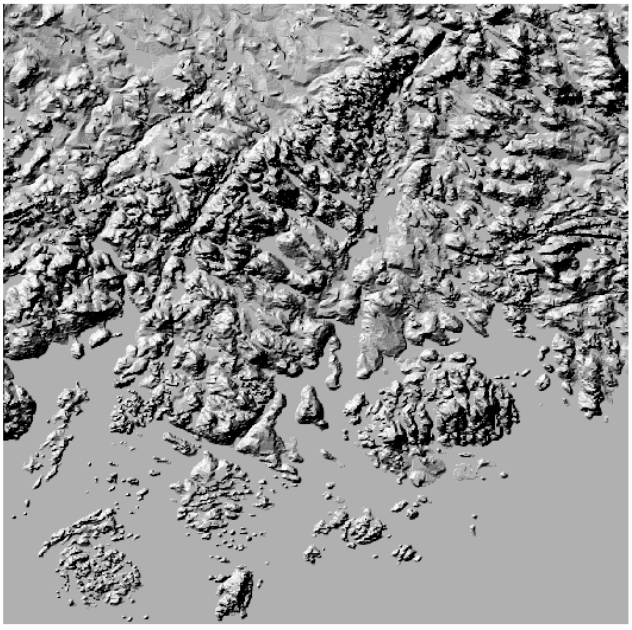


Figure 5: Shaded relief image of Maine, USGS Bangor 1:250,000-scale quadrangle

6 Metrics and Tactical decision aids

6.1 Phase 1 TDAs, metrics and applications: summary

In Phase 1 (but not Phase 2) these were designed to be generated interactively, rather that stored, and were as follows:

6.1.1 Tool to extract one visibility grid

java –cp CIDclass.jar uk.ac.ed.geo.cid.ExtractMAP <cid_database_name> <output_file_name> <row> <column>

The class to support this tool reads the parameters from the command line, creates an instance of the CID database class, reads the header information to initialize this class and then loads the MAP corresponding to the specified row and column from the database and invokes the write method to create a BIL file. (A function to write a text file also exists.)

6.1.2 Cumulative visibility

Result is a c x r grid of counts (`cumvis`) as shown in Figure 6.

The operation is illustrated in Figure 7 and in the following code:

```
Initialize cumvis to 0.
for each MAP(i,j)
for each row r
  for each col c
    if  (MAP [c, r] == 1)
     cumvis [c, r] = cumvis [c, r] + 1
```

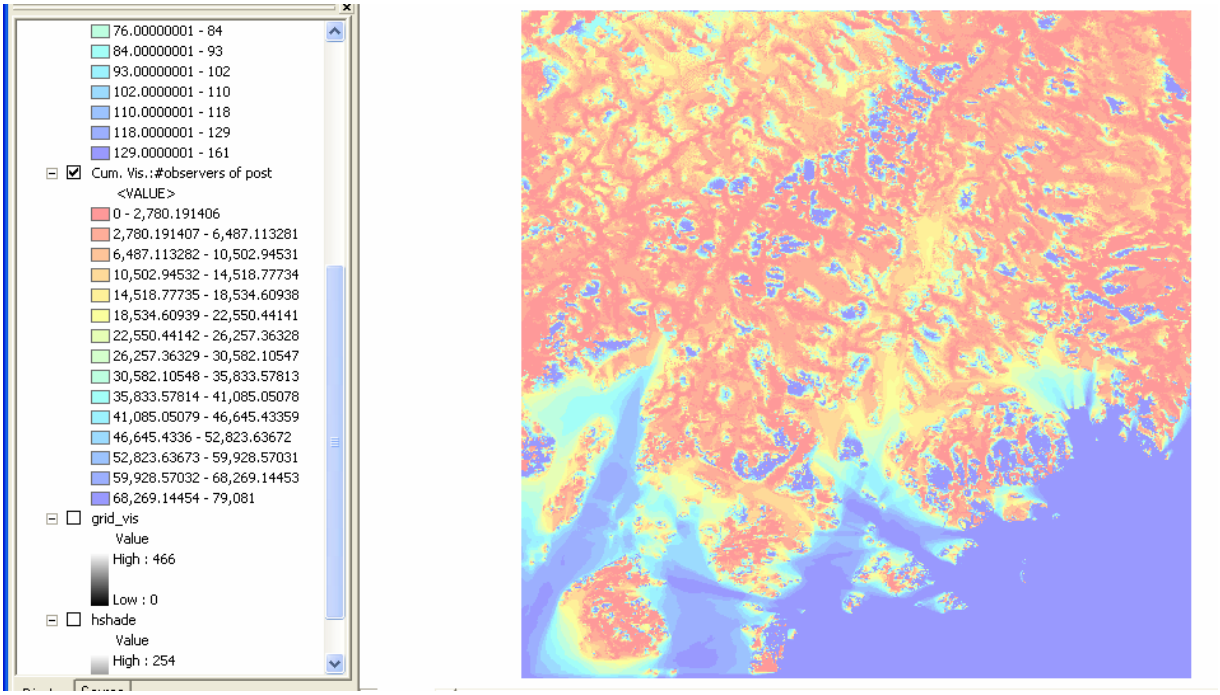


Figure 6: Cumulative Visibility (Observed)

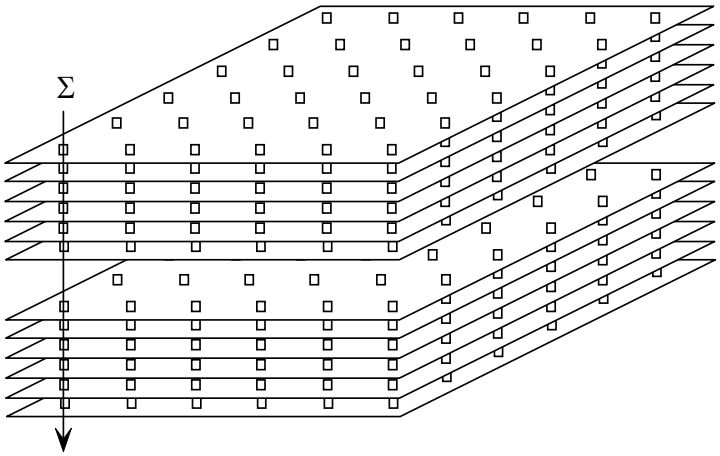


Figure 7: Cumulative Visibility Operation

6.1.3 Post of maximum visibility

Result is a c x r array of tables each containing a list of `col`, r`ow` and `count` values as illustrated in Figure 8.

This structure allows the case where more than one post has the same level of visibility from the specified post. The operation is illustrated in Figure 9 and in the following pseudo-code.

For		MVP is		
Row	Col	Row	Col	Visibility
65	23	129	268	57932
65	24	191	138	55445
65	25	42	45	21492
65	26	15	6	24838
65	27	15	6	24838
65	28	15	6	24838
65	29	15	6	24838
65	30	15	6	24838
65	31	15	6	24838
65	32	15	6	24838

Figure 8: Sample from results of a Maximum Visibility Operation

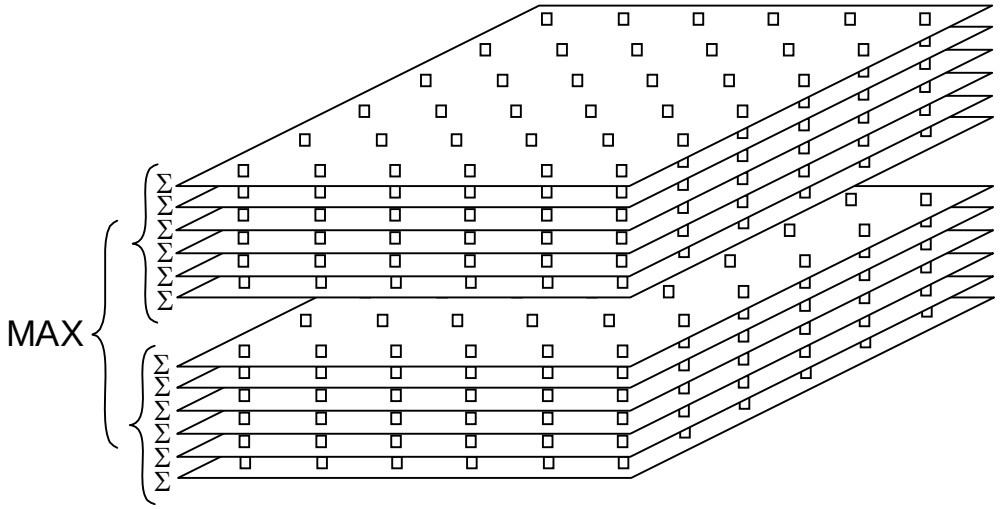


Figure 9: Maximum Visibility Operation

```
Initialize array of tables to null.
For each MAP(i,j) do
Calculate total visibility of MAP
sum = 0;
for each row l
  for each col k
    if (MAP [k, l]) == 1
      sum = sum + 1

for each row l
  for each col k
    if MAP [k, l] == 1  /* to test is post is visible in this MAP */
      if sum > count [k, l /* does this MAP have higher visibility */
      {
         add (i, j, sum) to table of values for cell [k, l]
      }
      else if (sum == count[k, l])
      {
         initialize the table for cell [k, l] to (i, j, sum)
      }
```

6.1.4 Multi-Observer Masked Area Plot

Result is grid of counts, c as illustrated in Figure 10b.

Input is a binary grid mask derived from a polygon or line, 1 defines a cell in the target.

The operation is illustrated in Figure 11 and in the following pseudo-code:

```
For each MAP(i,j)  do
sum = 0;
for each row l
  for each col k
    if (MAP [k, l] == 1  && poly [k, l] == 1)
                      /* post is visible and within polygon */
     sum = sum + 1
c[i, j] = sum
```

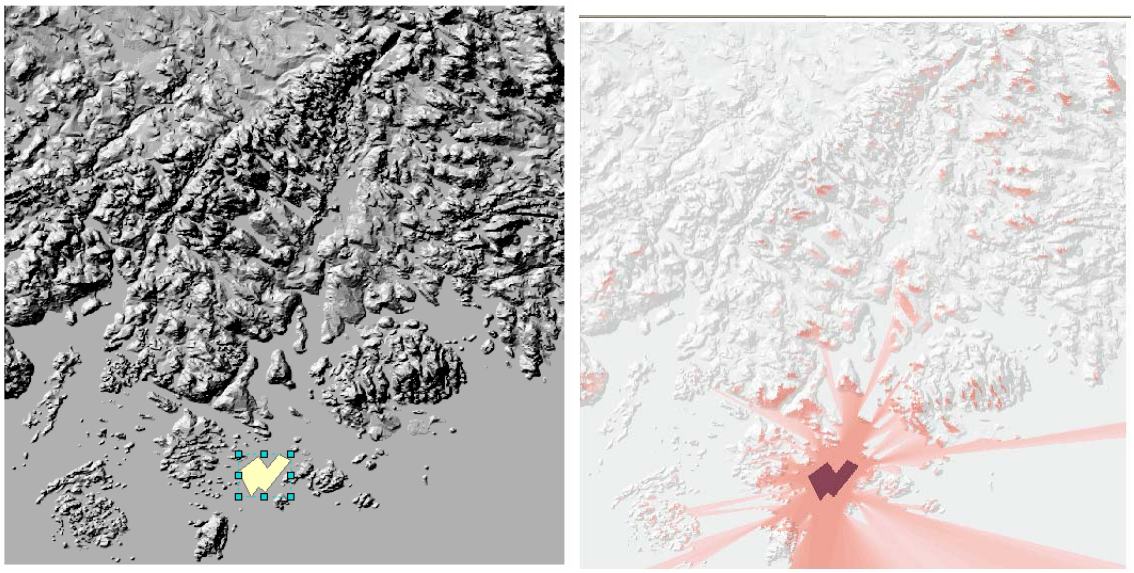


Figure 10: a) Input graphic defining target polygon; b) output visibility data showing mask of target

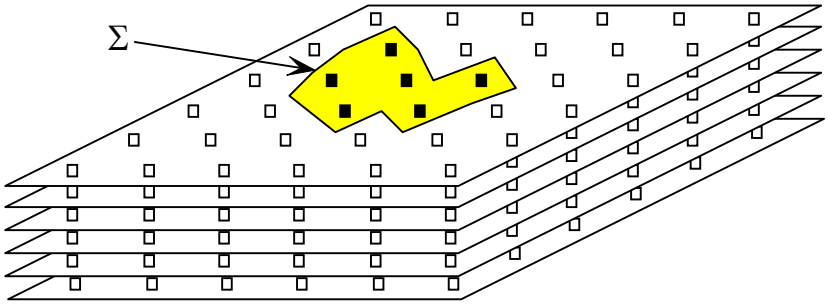


Figure 11: Multi-observer masked area plot

6.2 Overview of Phase 2 extensions

The Phase 1 Java applications were extended to generate descriptive surface metrics and interactive TDAs.

6.2.1 Metrics

The metrics are derived once, and held as part of the CID and were specified as follows:

1. Cumulative visibility (The number of posts observable from a post, derived by counting the bits in the observer's MAP, for all posts)
2. Core Area (Contiguous area around observer location) number of posts (Both 4 and 8 connectivity)
3. Fragmentation (Number of groups of contiguous areas making up MAP) (Both 4 and 8 connectivity)

4. Observer-neighbor visibility metrics. Several metrics are derived, each from a different mode of combination of the MAPSs for each post and its neighbors.

The design of the software recognized the benefit of being able to see the data from which a metric is derived – for example, the cumulative visibility (observer) metric at (i,j) is derived from a MAP for (i,j) – and so additional interactive display capabilities were developed for:

1) Display of a MAP with fragments colored differently (as a BIL file with pixel values being the fragment identifier). This also displays the core area.
2) Display of logical-MAPS (LMAP) – A temporary MAP derived by combining MAPs for (i,j) and its neighbors according to a logical function.

These were incorporated into the same user interface window used for MAP selection and display. Amongst advantages, these displays simplified testing, and should be an aid to building users' confidence in the system.

6.2.2 Interactive decision aids

The following were developed:

1) Target visibility. Based on the Phase 1 "multi-observer masked area plot", the Phase 2 development was primarily in the ArcMap extensions (section 7) to allow the user to define a target and derive a plot of the visibility of that target from each post in the DEM.
2) Route animation. This allowed a user to define a route and number of points along the route, and to see MAPs at each of these points. The displayed ArcMap can be stored as a JPEG.
3) Point of Maximum Visibility This derives a table for each (i,j) giving the location (C,R) of the point that can see the largest number of posts including this (i,j). Each table entry includes i,j,C,R and the cumulative visibility at that (C,R). This is displayed in an ArcMap table.

6.3 Phase 2 Algorithms

6.3.1 Metrics from 4 and 8 connectivity

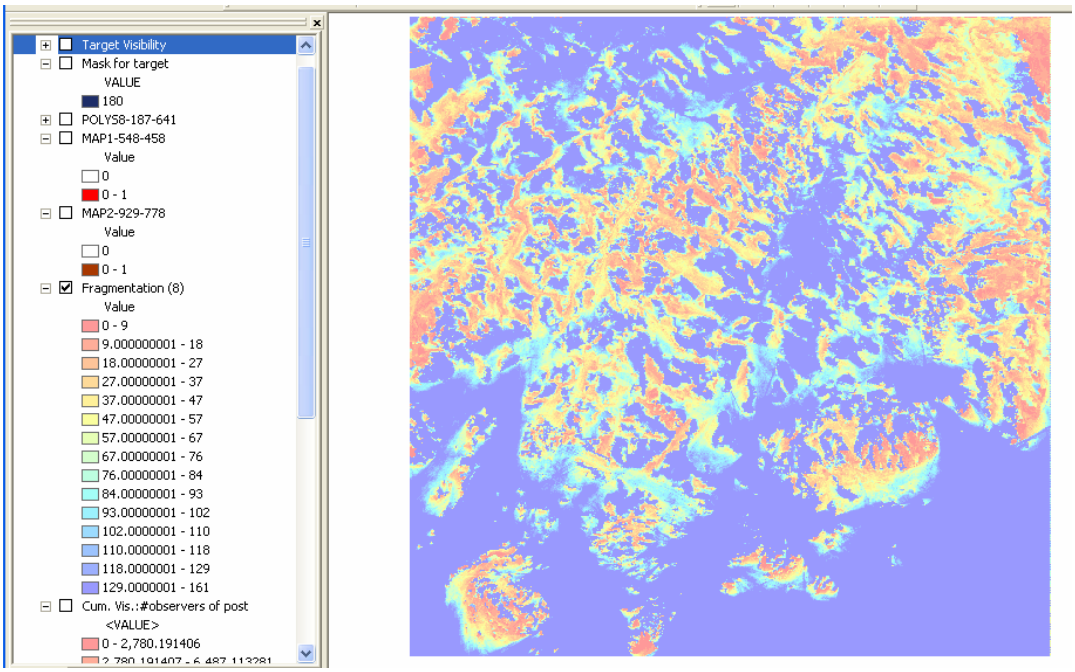


Figure 12: Fragmentation (8 connectivity): number of fragments

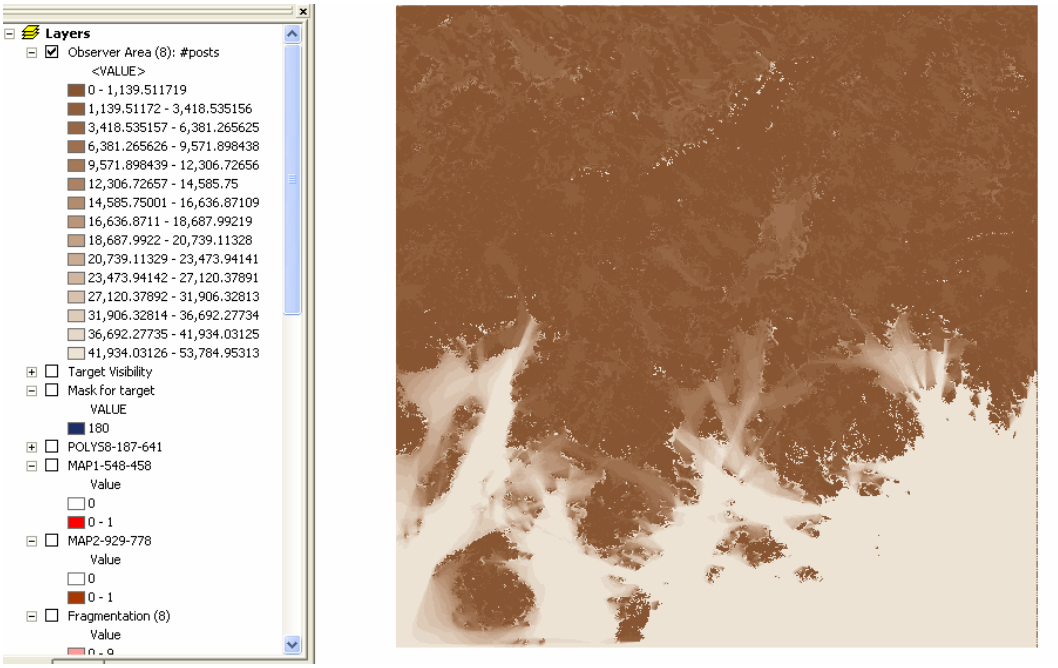


Figure 13: Core area (8 connectivity)

The fragmentation and core area metrics are derived together, separate runs determining the metrics for 4 and 8 connectivity. (The latter includes diagonal neighbours.) The algorithm was as follows:

```
Copy a MAP into an integer raster, setting cells to -1 if
the corresponding MAP cell is 1.
Seed fill from the observer location for polygon identifier
(id) 1
For each row
  For each col
     If pixel = -1
      Allocate a new id
      Seedfill from this location
     End if
  End for
End for
```

The seedfill function is a recursive algorithm to fill the complete span of contiguous cells in the row with the specified polygon id, scanning to both sides of the seed point, and then, based on connectivity (4 or 8) to set the limits of the span to be tested on both adjoining lines. For the spans in upper and lower adjoining lines in turn: if any adjoining cell is -1, then the seedfill function is called again.

By summing the number of pixels in each polygon (i.e. fragment) the observer's core area can also be derived: it always has a polygon id of 1. The number of fragments per MAP and the core area of the observer's fragment are written as BIL files, creating two of the required metrics. Generating the number of pixels in each polygon allows easy generation of the following metrics, beyond the specification:

- A BIL holding the fragment number of the largest fragment in the MAP. The key to its use is that this is 1 if the largest contiguous area in a MAP contains the observer; uses of this raster are likely to be in logical operations (if not 1 the precise value has limited usefulness beyond use in testing or in ArcMap to select the cells of this area.)
- area of the largest fragment (Equal to the core area if the observer is in the largest fragment)

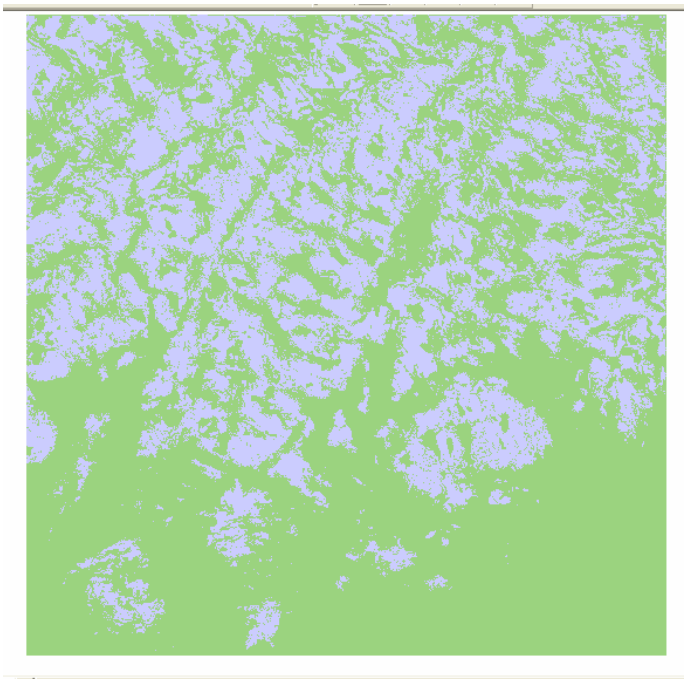


Figure 14: "Largest-area-pixel-id": Green where the MAP has its largest fragment contiguous with the observer.

Derivation of perimeters (perhaps the total in the MAP, in the observer's fragment and in the largest fragment) was recognized to be a useful extension, but one outside the timescales of the project. It could easily be added to the fragmentation code.

The implementation was revised for the larger dataset, for in analyzing MAPs such as at row 777, column 927 the code recursed 2132 times, more than was supported by the Java Virtual Machine on Windows. (The –XSs switch seems to make no difference – it should extend the JVM's stack.) To avoid this problem, the Java class "Stack" was used, to support push and pop operations. In place of the recursive call from/to SeedFill a push was made, and after the first call to SeedFill the Stack was emptied, entry by entry with an additional call to SeedFill (and further extensions to the Stack being made in place of recursive calls).

6.3.2 LogicalMAPs: observer and neighbors metrics

The "LogicalMAP" functionality evolved from the requirement to compare the MAP at a point (i,j) with its 8 neighbours and compute 1) the number of common points, 2) the number visible from (i,j) but not the neighbours, and 3) the number visible from the neighbors but not from (i,j). This involved comparing the MAP at (i,j) with the MAPs of its neighbors and to derive counts that are then written to (i,j) in the resulting grid.

A "Logical MAP" is the intermediate result of the comparison of MAPs. The metric derived from the LogicalMAP is a count of the number of set bits (identical to cumulative visibility for the "raw" MAP).

During implementation it was recognized that:

1) By formulating the required operations as logical functions a range of additional metrics could be provided.

2) Derivation of the required metrics could be coded so as to allow display of a single LogicalMAP for a chosen (i,j).

3) The above extensions would provide means for users to demonstrate the correctness of the metrics, as well as offering additional information.

Metrics are counts of the number of bits set in a LogicalMAP created from logical map algebra operations across 9 MAPs. Thus the metric at (c,r) is the count of bits in a logicalMAP, derived from centreMAP = MAP(c,r), (i.e., the MAP for row r, column c) and 8 neighboring MAPs (termed a haloMAPs or halo for brevity): MAP(c-1,r), MAP(c+1,r) etc.

The value of the metric pixel is thus:

logicalMAPpixel(c,r) = $\sum$ f (centreMAPpixel(i,j), { haloMAPpixel(i,j) })

where f is a logical operation and the sum is over all pixels (i,j)

The operations are:

Logical function	**logicalMAPpixel(i,j) = 1 if at (I,j)**	**Metric(c,r)**	***Metric at (221,3)***
or(halo)	any haloMAP is 1	Visible from any of the neighbors	3929
or(centre, halo)	centre or haloMAP' MAPs is 1	Visible from any of the neighbors or the centre.	3941
and(halo)	all haloMAP are 1	Number of points common to the halo	531
and(centre, halo)	all haloMAP and the centre are 1	number of common points	530
and(centre,not(or(halo)))	centre and none of halo is 1	Number visible from (c,r) but not from neighbors	12
and(not(centre),and(halo))	All haloMAP are 1, centre is 0	Number seen from every neighbor but not seen in the (c,r) pixel itself.	1
and(not(centre),or(halo)	Any haloMAP is 1, but centre is 0	Number seen from any neighbor but not seen in the (c,r) pixel itself.	2099

Table 1: LogicalMAP operations for observer-neighborhood metrics; example values with 8 connectivity are given for location row 3 column 221, from the first trial dataset, where cum.vis.(#posts seen) = 1842

Metrics are derived for both 4 and 8 connectivity, the former being most useful in testing. Testing included the following checks, the data being those from Table 1 and from corresponding 4-connectivity metrics:

a) internal consistency:
    a. or(halo4) <= or(halo4,centre) <=or(halo8,centre) (3226< 3245 < 3941)
    b. or(halo4) <= or(halo8) <=or(halo8,centre) (3226 < 3929< 3941)
    c. or(halo8) = or(centre Halo8) - and(centre not(or(halo8))) (3929 =3941 – 12)
    d. or(halo4) = or(centre Halo4) - and(centre not(or(halo4))) (3226 =3245 – 19)
    e. and(centre halo8) = and(halo8) – and(not(centre) and(halo8)): (530 =531–1)
    f. and(centre halo4) = and(halo4) – and(not(centre) and(halo4)): (857=858 -1)
b) consistency with cumvis observed
    a. cumvis = or(centre,halo8) – and(not(centre),or halo8) (1842 = 3941 – 2099)
    b. cumvis = or(centre,halo4) – and(not(centre),or halo4) (1842 = 3245 – 1403)
c) consistency with table of values from maps generated by logical operations
d) consistency with results from raster calculator and MAPs.

(Note that the boundary cells cannot conform to these rules, when testing in the ArcMap raster calculator.)

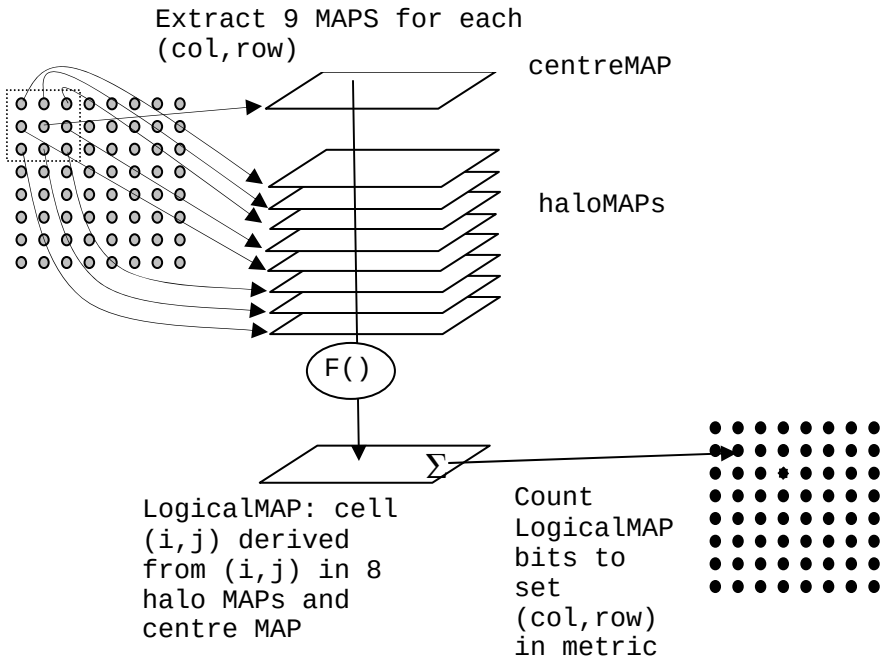


Figure 15: Derivation of Observer-Neighbor Metrics

The selected metrics are coded explicitly in the class LogicalMetrics, function metricate; the design was intended to facilitate the addition of code for further metrics in future.

6.3.3 Approaches to shape and orientation analyses

This section reports on a task to explore, but not implement, possible approaches for creating metrics that are related to shape and orientation.

In examples seen to date, MAPs are usually very fragmented and irregular and far from elliptical. Often the observer is not in the fragment of the map with maximum contiguous area – this was a motivation for the extensions described in 6.2.1. This can be seen in the Figure 16, showing a fragmented MAP. The red dot, highlighted by the MS Word oval graphic shows the observer location, the green fragment is evidently the largest.

The following alternatives were considered:

1. Determine and characterize the convex hull round a MAP.
    a. area and perimeter of hull
    b. total area and perimeter of contiguous groups in the MAP
    c. ratios between these
2. Filter MAP to remove scattered small fragments and then do something like the above – and compare fragmentation (number of contiguous groups) of filtered and unfiltered image. (Filter might be maximum likelihood: when a central pixel is a minority in a 3x3 window, change its value. Iterate to near stability or for a set no. of iterations.)

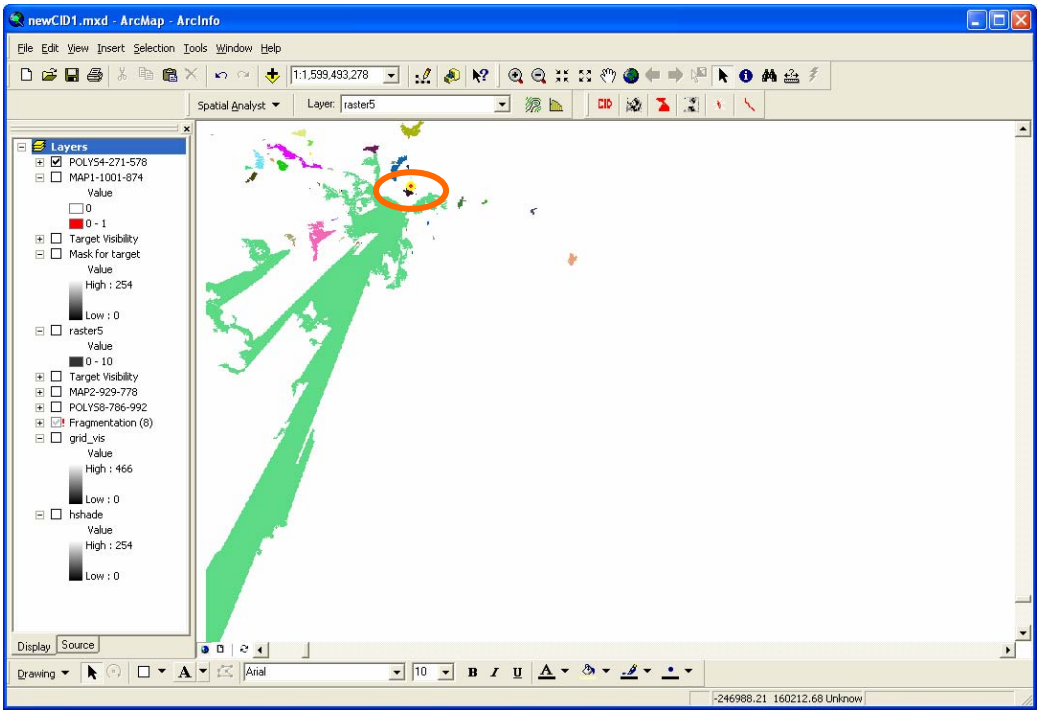


Figure 16: Example of a fragmented MAP: observer shown by red dot, is far from the centre of the visible posts; the largest contiguous area (in green) is not connected to the observer's post.

3. Derive the variation in density of the bits within the convex hull – e.g. take a moving window, for each post in a MAP derive statistics for number of bits that are set in the window: This seems is similar to concepts of "entropy" of shape – further exploration of these might be worth-while It is unclear how to characterize the distribution to create a metric – maybe as the number of moving-window locations in the hull of the MAP for which the density is under (perhaps) 50% - and whether this would be much different to information derived from the perimeter:area measures.
4. http://www.lpi.usra.edu/meetings/lpsc2002/pdf/2000.pdf describes methods for analyzing cross-sections of rocks (which are less fragmented than a MAP!). This leads to:
    a. fit an ellipse and measure error from that ellipse
    b. circularity: how circular is the MAP? – the circle centered on the observer
    c. angularity – a definition of this is not included in above html
5. Locate the centre of gravity (CoG) of the MAP, and its distance from the observer. Use the CoG as the centre of circularity measures

Perhaps reasonable first attempts would be:

- Shape: Derive hull and MAP measures of perimeter and area and circularity, where circularity is determined by the radius of the circle centered on the observer which gives the best fit, and a measure of goodness of fit between the circle and all MAP bits.
- Orientation: derived from a fit of an ellipse to the data to obtain the angle of the major axis.

6.4 Implementation - Java classes

Most of these classes were developed for the Phase 1 system, and extended in the current second phase.

6.4.1 Classes to represent a single or group of MAPs

A generic class *MAPBase* contains fields and methods to store a set of MAPs. The internal representation of a set of MAPs is as an array of bytes with a group of consecutive elements in the array representing the MAPs for a rectangular block of posts in the DEM. Static class fields *bitsPerCell* and *bytesPerCell* define the number of bits (and therefore posts) in the group and the number of bytes required to store the group. The default values are 16 and 2 respectively. The class also contains methods to read and write MAPs in GRIDASCII, binary (BIL) and compressed binary (zip) format. Methods to support calculation of the TDAs for a set of MAPs are also provided.

Class *MAP16* provides some optimizations for the case where a block of 4x4 MAPs are stored together. In particular there are optimized methods to support the three TDAs described in the project specification. The optimization takes advantage of the internal representation as a simple array of bytes and processes the data with a single loop, avoiding the overhead of nested loops. The manipulation of single bits is carried out using the & (and) operator where necessary, although the cumulative visibility calculation uses an array to convert the particular bit pattern in a byte into the number of bits set in that pattern as a short cut.

Class *MAP1* is the simple case of a single MAP, although in this case we use one byte to store a cell rather than using a single bit.

Class *MAPInt* is a specialization of *MAPBase* which is intended to store a DEM rather than a MAP but it takes advantage of some of the methods provided in the base class.

6.4.2 Classes to represent a CID (complete intervisibility database)

A generic class *CIDBase* contains fields and methods to manage access to a complete intervisibility database stored on disk and the allow operations to be performed on the complete database as well as tools to extract particular MAPs from the database. The methods *preProcessMAP*, *processAllMAP* and *postProcessMAP* allow collective operations to be carried out on the database. The methods *processAllMap* steps through all MAP16s in the database and calls *processMAP* on each in turn. Method *processMAP* is a dummy method intended to be overridden in subclasses implementing particular algorithms.

Class *CIDAreaVis* overrides *processMAP* to call the *polygonMaskedAreaPlot* method on a MAP.

Class *CIDCumVis* overrides *processMAP* to call the *CumulativeVisibility* method on a MAP.

Class *CIDMaxVis* overrides *processMAP* to call the *MaximumVisibility* method on a MAP.

Class *CountXY* provides the fields necessary to store the table of row, column and count values at each post in a maximum visibility analysis and is used in *CIDMaxVis*.

6.4.3 Applications classes to support command line utilities

Metrics are derived and TDAs used (Section 7) by invoking Java classes from the command line.

Target-visible TDA

Class *TestCIDAreaVis* reads parameters from the command line and creates an instance of *CIDAreaVis*, which is used to read and process a Multi-Observer Masked Area Plot operation. The result of the operation is written out as a grid in GRIDASCII format. The polygon to be used as the mask is specified in the <polygon_name> parameter and is expected to be a grid of the same size as the CID and in ArcInfo GRIDASCII format. The application is invoked from the command line as follows:

java –cp CIDclass.jar uk.ac.ed.geo.cid.TestCIDAreaVis <cid_database_name> <polygon_name> <output_file_name>

Cumulative visibility metric

Class *TestCIDCumVis* reads parameters from the command line and creates an instance of *CIDCumVis*, which is used to read and process a Cumulative Visibility operation. The result of the operation is written out as a grid in GRIDASCII format. The application is invoked from the command line as follows:

java –cp CIDclass.jar uk.ac.ed.geo.cid.TestCIDCumVis <cid_database_name> <output_file_name>

Table of maximum visibility

Class *TestCIDMaxVis* reads parameters from the command line and creates an instance of *CIDMaxVis*, which is used to read and process a Maximum Visibility operation. The result of the operation is written out as a table of 5 values with one line per row in ASCII format. The application is invoked from the command line as follows:

java –cp CIDclass.jar uk.ac.ed.geo.cid.TestCIDMaxVis <cid_database_folder> <output_file_folder>

and the "output file folder" is the metrics subdirectory in the CID database, to which is written the file *maxvis.csv*. (This must be loaded using the metrics tool of the CID toolbar in ArcMap (section 7) to an Arc table.)

java –Xmx100000000 –cp CIDclass.jar uk.ac.ed.ac.geo.cid.TestCIDMaxVis .\database .\database\metrics

The command can also be invoked by:

src\maxvis.csv.bat <cid_database_name>

MAP extraction from a CID

Class *ExtractMAP* reads parameters from the command line and creates an instance of *CIDBase*, which is used to read a database and extract the MAP for the post at a particular row and column. The result of the operation is written out as a grid in BIL format. The application is invoked from the command line as follows:

java –cp CIDclass.jar uk.ac.ed.geo.cid.ExtractMAP <cid_database_name> <output_file_name> <row> <column>

MAP fragmentation metrics

Class *TestCIDFragments* reads parameters from the command line and creates an instance of *CIDBase* which is used to read a database and extract the MAP for each post. The result of the CIDFragments operation is written out as a grid in BIL format. The application is invoked from the command line as follows:

java –Xmx100000000 –cp CIDclass.jar uk.ac.ed.geo.cid.*TestCIDFragments* <cid_database_name> < connectivity >

where <connectivity> is 4 or 8.

Two additional parameters can be used: the first and last rows for which the metric is defined. This is primarily for testing purposes.

Logical MAP metrics

Class *makeMetrics* reads parameters from the command line and creates an instance of *LogicalMetrics* which is used to read a database and extract the MAP for each post. The result of the CIDFragments operation is written out as 14 grids (corresponding to different logical operations on MAPs) each in BIL format. The application is invoked from the command line as follows:

java –Xmx100000000 –cp CIDclass.jar uk.ac.ed.geo.cid.*makeMetrics* <cid_database_name>

MAP fragmentation

Class *FragmentMAP* reads parameters from the command line and creates an instance of *CIDBase*, which is used to read a database and extract the MAP for the post at a particular row and column. An integer BIL is derived, in which each visible cell holds the identifier of its fragment, fragment 1 including the post at (column, row). The result of the operation is written out as a grid in BIL format. The application is invoked from the command line as follows:

java –Xmx100000000 –cp CIDclass.jar uk.ac.ed.geo.cid.ExtractMAP <cid_database_name> <output_file_name> <row> <column> <connectivity>

where the final parameter is 4 or 8.

6.5 TDA Performance

6.5.1 Metrics

Table 2: Runtimes for generating metrics (1201x1201 dataset)

CSV file with max. visibility data	1.22hr
Cumulative visibility (observer)	0.9hr
Fragments (4 connectivity)	27hr
Fragments (8 connectivity)	27hr
Logical metrics	6days

Runtimes for creating LogicalMAP metrics are long: all such metrics are created in one run. (With the smaller dataset runtimes were 24hrs 14mins on the 300MHz Sun.)

With relatively minor developments, metric generation could itself use Condor. Condor does have support for Java classes; this has not been explored.

1. Distribute tasks, each deriving part of the metric for a subset of rows; this entails sending a) the relevant rows' zip files to the worker via condor submit and b) a batch file to invoke the Java classes. The Java classes would also need to be downloaded, to avoid use of networked disk drives (a current Condor constraint).

2. For each metric, each worker would write a GRIDASCII or BIL file with its data, into a file named according to the metric and the set of rows processed. Condor would return these files to the coordinating process.

3. A sink process would be coded to gather data from all tasks' files into the resultant metric files. This would execute on the coordinating processor.

The scope for optimizing the code has not been explored.

6.5.2 Interactive TDAs: Multi-Observer Masked Area Plot

For a CID based on a DEM of size 1201 by 1201, performance is not adequate for interactive use (~45minutes). At the outset of this project it was recognized that an "inverse" CID is needed, where each Inverse-MAP(i,j) holds those posts that can see the post (i,j). The number of inverse-MAPs to be accessed

would then be the number of posts in the target, whereas currently all MAPs have to be accessed. Creating an "inverse CID" was not part of the project specification and could not be accomplished in the time available.

7 ArcMap extensions

ArcMap is an application in the ArcGIS product suite produced by ESRI. It supports the display and analysis of GIS layers of data, and is itself based on the ArcObjects software. This uses COM objects with published interfaces, thereby allowing extensions to be developed to fit seamlessly into ArcMap.

This section describes the functionality and development of software tools for accessing metrics and using the TDAs. These were built into the ESRI ArcMap application, and invoked the Java described in a previous section (6.4), with the Java running on the same processor. Extensions to ArcMap have been written using Visual Basic (VB) with ArcObjects. The VB code runs a shell script that invokes the Java, and then uses a named pipe to signal completion of the Java program to the VB. The VB and Java exchange data in temporary files, the results being displayed in ArcMap.

The tools are of two types. The descriptive products derive data that are fixed: metrics and the maximum visibility (MV) table. The metrics and MV table are held as part of the CID database (in \metrics), but are only derived once, when first requested.

The second type, the interactive tools, process the CID using different data derived from user input: points (by mouse click), lines or polygons (from graphics or else feature layers). These interactive tools produce files that are overwritten each time the tool is re-run (to prevent generation of growing numbers of data files). These are held in a "cidtemp" folder, created at installation time.

Development was undertaken using VB, as a) more relevant code already existed in VB than in alternatives, b) the only viable alternative in Edinburgh was C++, and c) prototyping in Visual Basic for Applications (VBA) in ArcMap as macros is an effective way to proceed and is not available for C++.

7.1 Tools for Metrics and TDAs

7.1.1 The CID Toolbar

The toolbar and associated code are held in the Dynamic Linked Library (DLL) CID1.DLL, which must be linked to ArcMap (Tools, Customise, Toolbar, add from file)

Left to right, the tools are:

1. CID initialization (only this is enabled when ArcMap is started)
2. Add CID metrics
3. CID Information
4. Clear all graphics symbols
5. Target Posts visible
6. Animation of route MAPs display

7.1.2 CID initialization

Before a CID can be accessed, the user must have the DEM displayed in ArcMap, with the map being in the projection and units of the DEM (otherwise results of clicks and displays are uncertain). The initialization tool has two dialogues: the first to find the top directory of the CID database and the second to identify the layer comprising the DEM

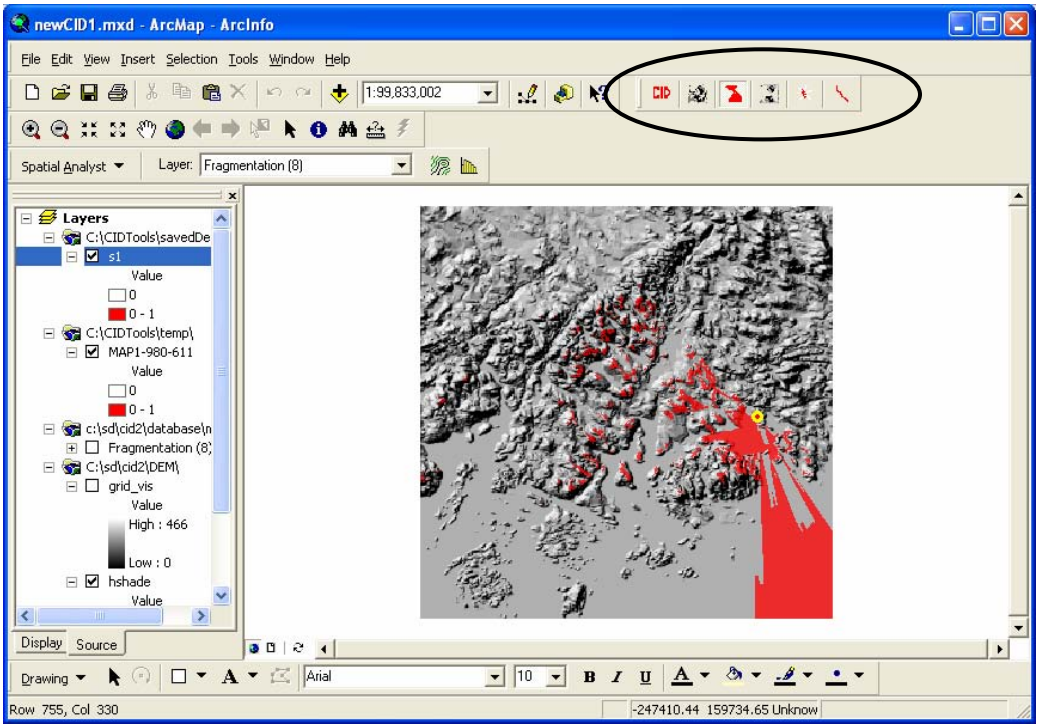


Figure 17: ArcMap window showing CID toolbar

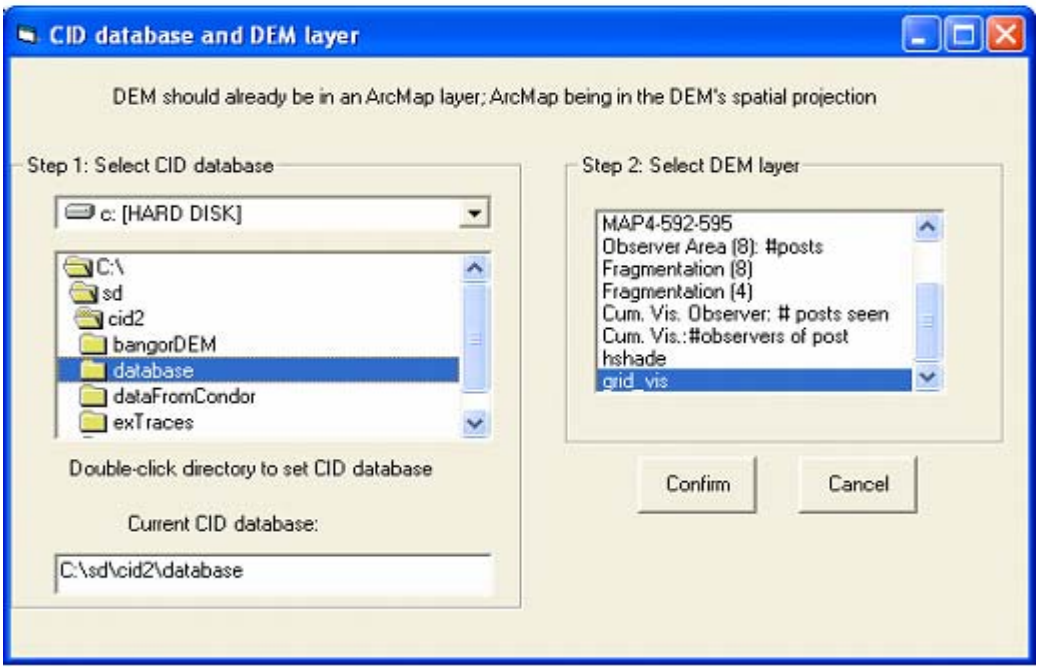


Figure 18: CID Initialisation

7.1.3 Add CID Metrics

There are 25 metrics in total: cumulative visibility (observed and observer), the table of maximum visibility, 4 for connectivity with 4 neighbors; 4 for connectivity with 8 neighbors, 7 for logical metrics with 8 neighbors, and 7 for logical metrics with 4 neighbors. On loading the descriptor tool, a list box is populated with prompts taken from the list in file %CIDTools%\etc\ CIDDescrPdts.txt. The user can select one or more descriptors from the list. The display of the corresponding layers uses these strings as layernames in the ArcMap table of contents, applied to layer files taken from the *database\metrics\layers* folder.

The layer files can be manually updated to correspond to the specific CID's metric: it is for this reason that they are held in the database directory tree, as described earlier.

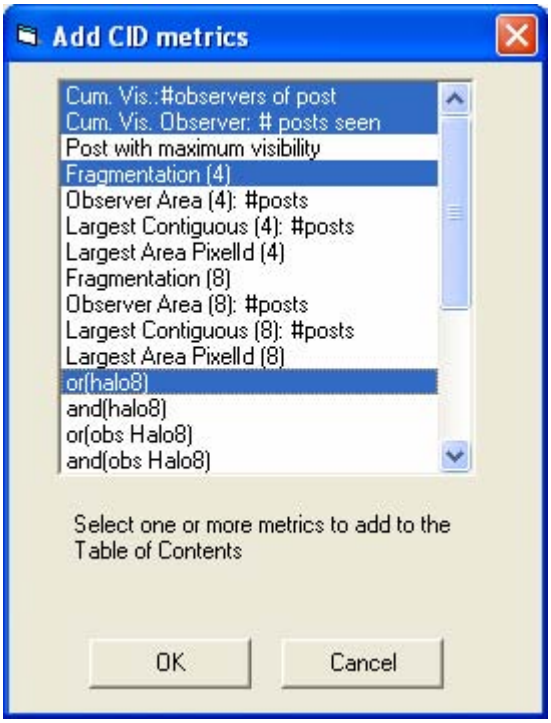


Figure 19: Dialogue to add metric layers and the Maximum Visibility Table to ArcMap

7.1.4 CID Information

This tool combines the following requirements:

1. Display a MAP: Using the row, column pair; the x, y coordinate pair, or the cursor; extract a MAP from the CID and create a layer in the Table of Contents displaying the MAP in red with 50% transparency.

2. Display descriptors (metrics and posts of maximum visibility)

On display of this form, the user is in "interactive" mode. Alternatives to interactive mode are to enter coordinates or else row/column selections.

In interactive mode, clicking on the ArcMap, in the region of the DEM will cause the corresponding MAP to be displayed

The list box in the input dialogue also allows selection of alternatives to the MAP – these being the MAP of the post with maximum visibility including the selected post (if multiple such posts exist, a warning message is given), a MAP colored by fragment (i.e. by contiguous regions of cells), and logicalMAPs; the list box entries are read from cidMAPS.txt in %CIDTools%\etc.

.

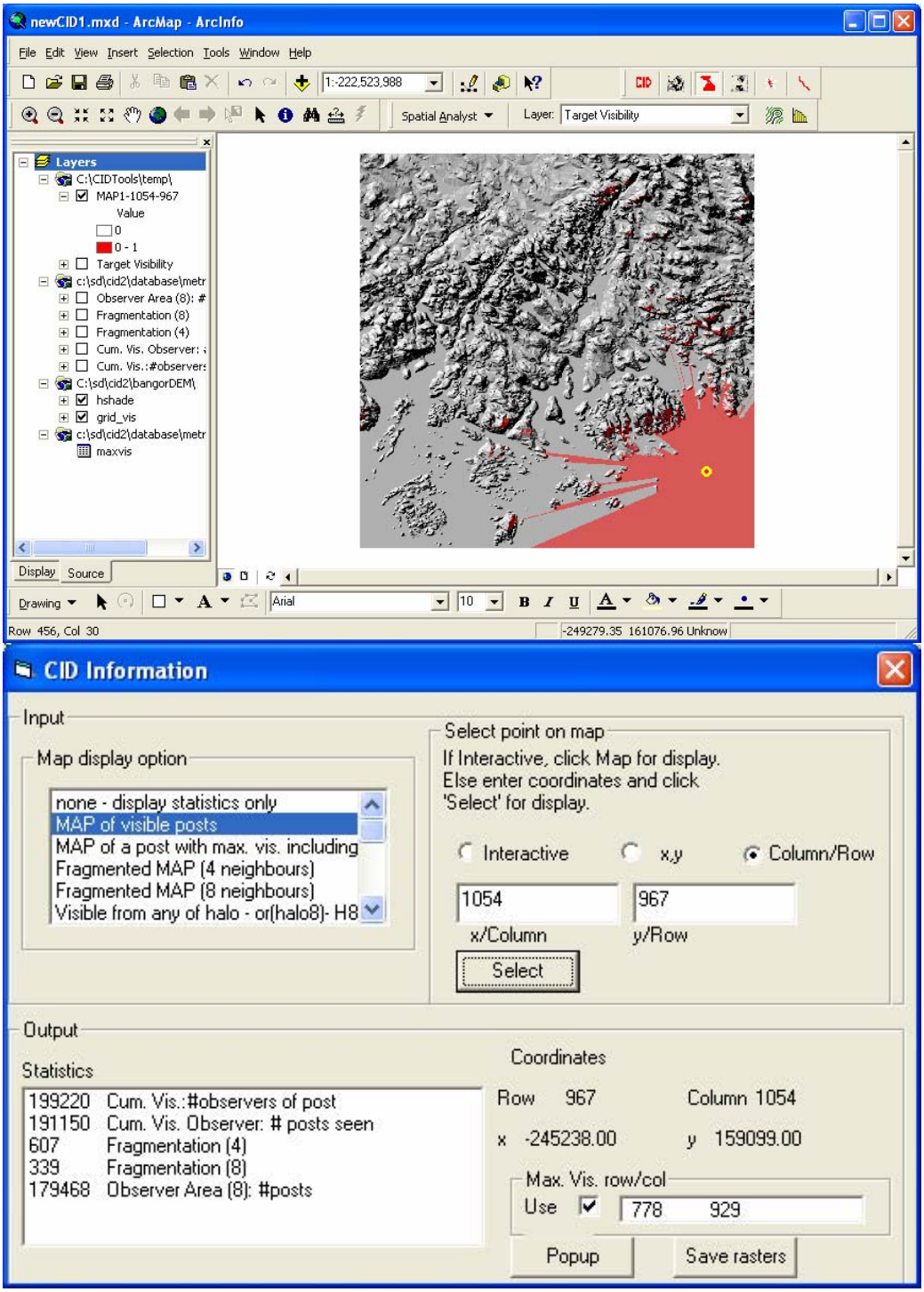


Figure 20: CID Information Dialog

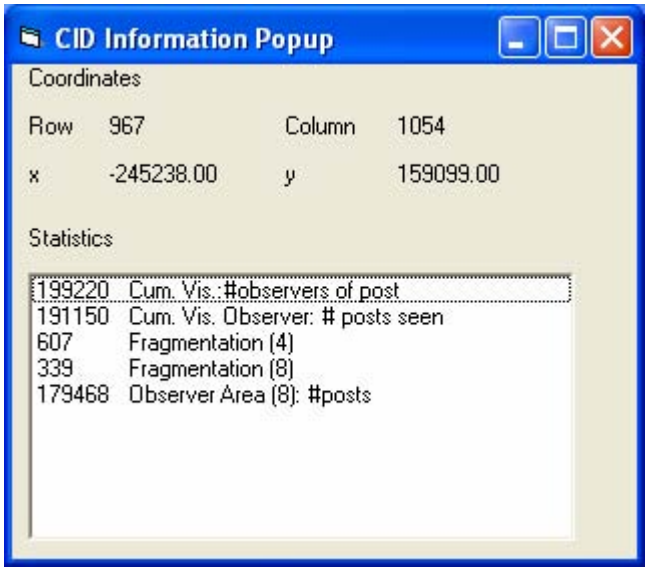
Up to four MAPs can be seen at any time in interactive mode – by clicking with shift, alt, or control, MAPs 2,3,4 are shown. This applies to logicalMaps and MAPs, but not to the multi-colored fragmentMaps) The ability to display multiple MAPs was not an initial requirement, but was thought to be useful.

When a MAP is displayed, metric data for the chosen post are listed in the window in the Output statistics list box. The length of this list box is determined by the layers in the ArcMap table of contents (whether or

not these are visible layers): if “fragmentation (4)” metric is in the Table of Contents then its value at the chosen post is listed

If the “Max. Vis. row/col” checkbox is set, then the table of maximum visibility is interrogated and a list of the points with maximum visibility, including that of the chosen post (usually but not necessarily a list of length 1) is output. In some cases this list is the row/column of the chosen post itself, for posts with high visibility. These list-box entries can be double-clicked to cause the MAP for the MVP to be displayed. The option, offered by the checkbox, optionally speeds access by avoiding the interrogation of the MV table.

The “popup” command opens a new window (modeless so that it remains on the screen until it is closed), with the statistics listed for the chosen post.



.

Figure 21: CID Information Popup

The “Save rasters” command in the CID Information window allows any of the rasters in the table of contents to be selected and copied to a new BIL file: this was provided to allow saving of the temporary layers (MAPs, LogicalMAPS, fragmented-MAPs, nposts visible in a target), which are otherwise overwritten by subsequent requests. Files are copied with their symbology, so if the user specifies “dest” as the target destination, then the files dest.bil, dest.lyr are created in the directory selected. The copied layer is displayed in ArcMap.

The CID Information window is modeless, so that other ArcMap commands can be interspersed with MAP selection. Choosing any other CID command will hide the CID Information window: reselecting the tool from the CID toolbar recovers the window with its previous content. Closing the window by clicking the X in the CID Information title bar loses that context, when the tool is next used.

To allow browsing without excessive use of local disk space, and to allow comparison of MAPs, 4 MAPS are displayable via BIL files at any time. The BIL files are temporary, being overwritten on the next command; each has a corresponding layer file that contains the symbology. The four MAPs will be displayed in different shades as defined in the layer files. It is here that the transparency and color of data are defined – and these are selected by keying CTL, SHIFT or ALT when clicking in interactive mode. (This keyboard option is not active for fragmented maps as these are multicolored.)

For example, clicking without any keyboard input will display map1.lyr with a new map1.bil displayed within it. When displayed the MAP layer name is MAP1_*row_col*, and a graphics symbol will show the observer location.

The ArcMap menu tick for fragment layers will sometimes be a pale gray, not the usual black. In this case right click on the layer name, and choose “Visible scale range” then “Set maximum scale” to see the

fragment colors. If ever there are missing polygons (as compared with the MAP) then also check the layer scale – in all tests to date such issues have been due to ArcMap visualization rather than algorithmic errors in fragmentation.

The menu for the CID Information list box of options for display is held in CIDMaps.txt in %CIDTOOLS%\etc.

7.1.5 Target Visible

The target visible TDA calculates the number of posts of a target feature that are visible from each point in the DEM. Point, line, or area target features can be modeled using a raster grid, where the raster grid is determined from:

1. A selected graphic (line or polygon described by vertices (neither ellipses nor circles have been successfully used due to an apparent problem in ArcObjects)),
2. A feature layer (all features are used to determine the raster) or
3. A raster layer (Non-zero, valid values are used in the raster to define the target)

The tool extracts the raster to define the target, using components of Spatial Analyst. Spatial Analyst must therefore be installed and running with its default cell size and raster size set to that of the DEM (Spatial Analyst – Options menu). The CID tool exports that raster as a GRIDASCII file to be used by the Java. The resultant BIL file gives the number of the target posts that are visible from each DEM post. Runtimes are of the order of 45 minutes for the 1201 by 1201 DEM.

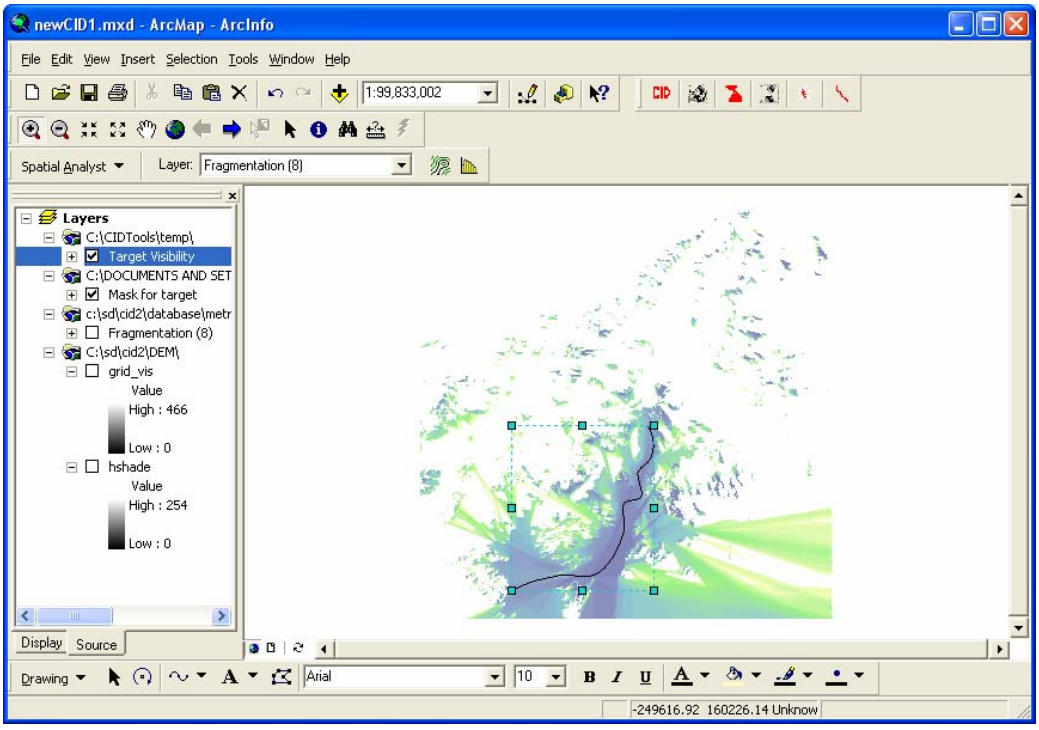


Figure 22: Target-visible TDA - example for a linear graphic

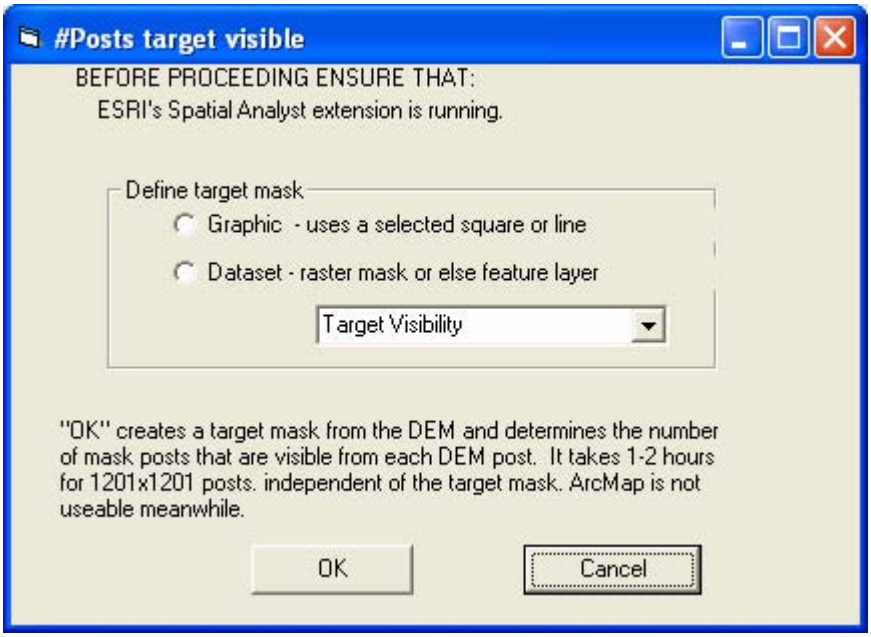


Figure 23: Target-visible dialogue

7.1.6 Route Animation

This tool displays in turn all MAPs along a route defined by a selected linear graphic. The animation routine steps through the MAPs along the route and optionally performs a screen capture at each point.

The user will define a map background in ArcMap; the MAP1.lyr file used in the CID Information dialogue defines the color and transparency level for the MAPs.

The user can determine the number of MAPs to be equally spaced along the graphic or alternatively to be all DEM cells that are intersected by the graphic. (Intersection is determined by use of an ArcObject method that returns the coordinates of a point at specified intervals along a line. If all DEM cells are required along the graphic, then intervals of $1/10^{th}$ cell are used – so intersections of under *0.1*cell dimension* might be missed.)

The screen captures are numbered <filename>0001.jpg, <filename>0002.jpg, and <filename>XXXX.jpg, so they can be automatically loaded in an animation program. The user will create a directory for the results, from a dialogue that will also determine the <filename>.

As is evident from the dialogue window shown, the modes of defining the MAPs to be displayed are:

1. Input the name of a file containing (x,y) coordinates (This could be created by a previous use of this tool, or by other software.)
2. Creation of a new file of x,y from a line in the graphics layer
3. Creation of a new file from vertices of a 1-D feature

The Display list box allows selection for display at each point of:

- The MAP of the point
- The MAP of post with Max visibility including route-post as a target.
- Both of the above
- Display of MAPS as fragment-4 and fragment-8 format, showing the regions of connectivity as one color

The interrupt button will stop the animation after displaying a MAP.

Points for which MAPS will be displayed are shown in yellow; those already displayed are in green.

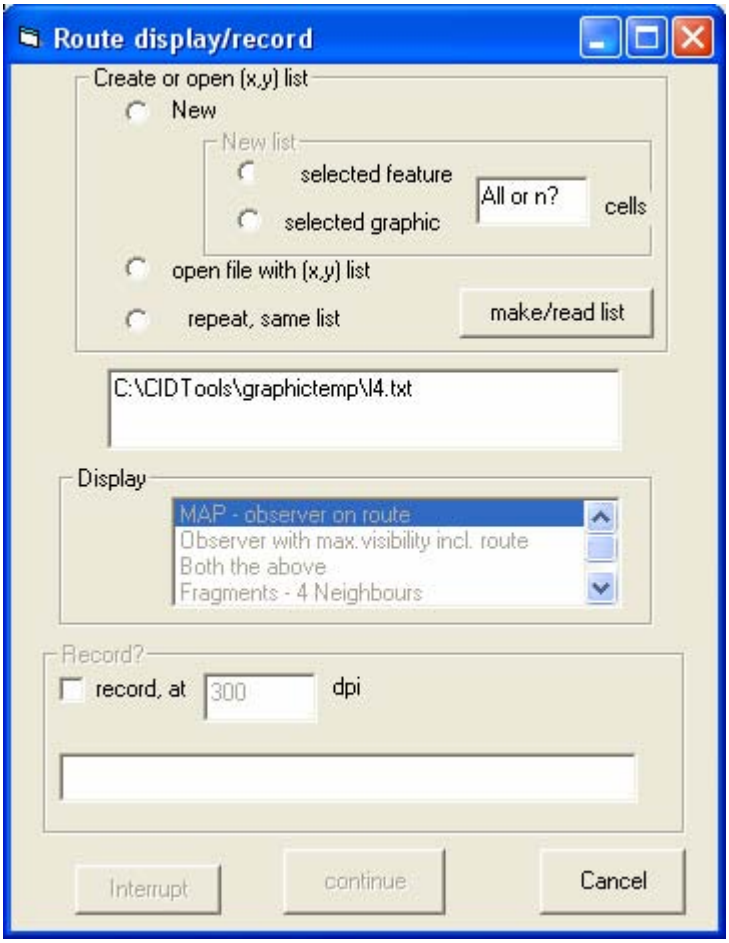


Figure 24: Dialogue form for the Animation Tool

Desirable extensions for future might include:

- Sum visibility along the route from all chosen posts. (Alternatively, do not clear one MAP before showing the next. This was not done, as the table of contents would grow very long for anything more than short lists of points.)
- Automatic panning to follow the route.
- After an interrupt, be able to restart from where reached (to allow reset of extent via ArcMap tools, for example)

7.2 Design aspects

7.2.1 CID access and TDAs: Use of Java from VB

The option of accessing the CID directly from an ArcMap extension was rejected. Java remains the interface to the CID:

a) If CID changes in design, then only one package of code must change (cf. if CID was interrogated from VB in ArcGIS)
b) Eliminates VB development of code that already functions in Java.
c) In strategic terms, this also leads towards the distribution of processing with the CID being remote and accessed via Web/Grid services

The Java-VBA interface uses a shell call from VBA. A shell call is non-blocking, needing an additional method to synchronize with Java results. A named pipe was used to notify completion of the Java from the shell script. It is passive, taking little or no CPU time during the Java execution. VB for pipes is at http://support.microsoft.com/default.aspx?scid=KB;en-us;q177696 and http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ipc/base/interprocess_communications.asp

Java's Virtual Machine starts sufficiently quickly that there is no need to run a Java umbrella program itself calling all other Java.

The current VBA-Java interface requires that the VBA validates inputs – e.g. row/columns being on the raster – as there is no status returned from the Java at present. The pipe was used to enable this status to be returned, but this level of functionality has yet to prove necessary, and so implementing this was de-prioritized i.e., the assumption is made that the Java works when given correct inputs. When issued, the Java command issued is held in %CIDTools%\temp\cidJava.bat. For cases where debugging proves necessary, e.g., if a command fails, then this should be run from a command window to see any statements written out from the Java.

7.2.2 Development environment: ArcMap and VBA

It was decided to prototype some code in macros in ArcMap, then to port to a DLL, as a DLL (in comparison to a VBA script in ArcMap):

a) Is easier to deliver and install
b) Gives a richer user interface
c) Is more consistent with ArcGIS approach – only trivial codes are usually found in macros.
d) Avoids need to specifically register any libraries used in the code.

Development in .NET was not an option, as the ArcGIS available at the University was version 8.2 until after development was completed.

Nuisance factors arose from difficulty of debugging – when building the DLL to start ArcMap and supposedly allow debugging, interaction with the map failed and any attempt to display a new layer caused both VB and ArcMap to hang. In these cases, when running without debug-mode, the code was fine. There is an indication of such problems on pages 124-5 of Exploring ArcObjects (EAO). The implication is that code modifications and testing are sometimes unsupported by a debugger. ArcObjects is brittle, with a tendency to disappear without trace. Attempts have been made to navigate around this – see "Known bugs" for the status of this.

ESRI is sometimes confusing in their discussion of VBA and VB; e.g., page 98 of EAO: by VBA they mean using VBA in ArcGIS, and by VB they mean using VBA in the Microsoft VB environment; the confusion is common, but still can be problematic!

7.2.3 File systems

The following directories are used:

%CIDTools%	To be defined by user at installation time. Its subdirectories contain all CID-tool-specific files.
%CIDTools%\etc	Files required by the VBA code in ArcMap: batch files to run Java, *CIDdescrPdts.txt*
%CIDTools%\src\	Source code for VBA, Java and MCA All Java class files are relative to here. e.g. uk.ac.ed.geo.cid.ExtractMAP is relative to this. If individual classes are used then the CLASSPATH should include this directory, otherwise the CLASSPATH should include the jar file containing the classes.
%CIDTEMP%	Defined at installation time. Location for files generated by the interactive tools.

The delivery system is set up for CIDTools to be C:\CIDTools and CIDTEMP to be C:\CIDTools\temp so all layer files are delivered assuming this to be the case.

The directories can be changed, but the layer files will need to be reset if a new CIDTEMP is used.

7.2.4 Layer management

Layers for display of metrics and MAPs are pre-created, held in the database\metrics\layers directory and will use defined file names: e.g. map1.bil for the first MAP. (Temporary .bil files are held in CIDTools\temp) These layers define transparency, colors, etc, and can be modified if wished. *Subsequent commands to display MAPs overwrite the BIL datafiles: saving a Map with these temporary files included will give unpredictable results.* A copy function (see CID Information tool) therefore is provided to allow data to be moved from selected layers to desired permanent locations.

In each CID database is a metrics directory and a metrics\layers directory. A consequence is that write-access is therefore needed to the metrics\layers folder… it would be preferable to have only read-access to all the database

7.2.5 Graphics layer

When MAPs are displayed, the observer is marked by a graphic, color coded to match the MAP color, which itself is determined by MAP number (1-4, there being up to 4 MAPs displayable at a time from the VBA code). The colors of the symbols are hard-coded.

When a subsequent MAPn is displayed, the previous symbol is found, deleted and a new symbol created. The matching of symbols is inelegant and on the basis of the green intensity.

It would be preferred to couple the graphic to the layer more strongly, which may be feasible, but was not achieved in the time allocated

8 Future directions

These can be divided into short term or tactical and longer term or strategic possibilities:

1. Tactical:
    a. Inverse CID: create and also hold an inverse CID: what can see each post, to enable interactive TDAs.
    b. Parallelization of metrics and TDAs,, including by use of Condor and Java capabilities.
2. Strategic: Grid-enable both CID creation and TDAs to allow remote easy access to high performance implementations.

9 Appendix: Obstacles encountered

The goal of this section is to summarize issues that had major impact on the project use of time: these were not unexpected, but are considered worth noting for the benefit of people extending these methods who are also new to them.

Windows has a mottled reputation, its development seemingly focused on the undemanding MS-Office and home multimedia user so that software developers struggle with some aspects, such as: remote procedure management, run time monitoring, and priority control. Condor provided a solution to these issues; only the way ArcInfo handled priority, as discussed above, proved problematical.

ArcObjects is a powerful concept that proved hard to work with, in the author's opinion, whilst emphasizing that the development was on version 8.2 of ArcGIS and improvements may well have occurred in 8.3, and are sure to do so in v.9. The difficulty arises from:

1) Documentation being automatically generated (apparently) and recursive: trial and error is sometimes needed to find the correct methods and properties and interfaces when documentation should have sufficed.
2) Examples suffer from the Goldilocks syndrome: some are too big and complex, some too narrow to show what is actually achieved, some don't exist, and not many are "just right", which would mean being simple to see what is happening, with a context that covers a range of uses.
3) Tools for validation of code are non-existent. There is need for a "debug mode" that tests the validity of every interface invoked. On countless occasions ArcMap failed with no trace or help provided; the limited means for debugging a DLL then exacerbated the problems and blood pressure. ("On error goto…" helps, and should of course be used, but does not clear this errant behavior in the author's view.)

The discovery of examples determines the development time needed, so the management of ArcObjects development is a fraught matter with very large uncertainties.

10 Appendix: Installation:

10.1 TDAs and ArcGIS software

Requires:

- Windows 32 operating system: XP has been used most often in testing.
- ArcGIS 8.2 or later. (The development was on 8.2; 8.3 has been used latterly.)
    - Requires ArcGIS Desktop with:
        - extensions - spatial analyst and 3D analyst.
        - Draw tools are needed for route and target specifications. (A routinely accessible toolbar in ArcMap.)
- cid1.dll – the CID tools written in this project
- Demonstration map in tecDemo.mxd in C:\CIDTemp

PCs that are used by the multi-computing system need both Desktop and Workstation (i.e. ArcInfo). When installing there is an option to allow command-line use of ArcInfo. This must be enabled to allow scripts to run the visibility analysis.

10.2 Programming languages

10.2.1 Java

JAVA_HOME set to the Java installation directory (…\bin)

CLASSPATH set to include the root of the CID Java applications – probably C:\CIDTools\src (the folder in which is found uk\), or the jar archive containing the CID classes, normally CIDclass.jar.

PATH to include JAVA_HOME.

10.2.2 Perl

On the MCA coordinating machine: V. 5.6.1 or later.

10.2.3 VBA

The DLL was developed in VBA using Visual Basic 6.

10.3 Environment Variables

The CID tools require, before ArcMap is run, that:

1. Two environment variables exist:

CIDTEMP set to C:\CIDTools\temp

CIDTools set to C:\CIDTools\

2. The JAVA_HOME and CLASSPATH environment variables must be set, with CLASSPATH including the src directory, in which is the uk\ directory of CID's Java.

3. cid1.dll is linked into ArcMap (In ArcMap select Tools-Customize-Toolbar-Select From File and browse to cid1.dll; drag new CID symbols onto the toolbar of ArcMap)

11 Appendix: Known bugs and eccentricities

The following have occurred on v 8.3 of ArcMap with Service pack 2 installed:

1) When displaying a fragmented map, not all fragments are visible (recognizable by displaying both the MAP and the fragmented-MAP). This is often fixed by changes to symbology, to be made by the user; these can be held in the lyr file. However, above an undetermined number of classes, the symbology with unique classes fails as follows:

   a) Right-click on properties, choose symbology, unique values, add all values. When commit to the change, ArcMap crashes. (On 8.2 this was also seen.)

   b) After the remove on the Window, the 0 is sometimes colored in when it should not be so. (A small price to pay for random cases where it doesn't crash.)

   Possible work-arounds:

      i) On the assumption that the 500+ unique classes sometimes needed are breaking an ArcMap unprotected limit to the number of classes, only use 1-128 (or 256? or 512?) for fragment displays, and risk adjoining polygons having the same value. This would lose the number of polygons in the individual fragmented maps, a useful check on the fragmentation metrics.

      ii) To avoid further coding and to maintain the information of unique fragment ids, allow ArcMap to display a subset of the polygons (if unique classes are used for determining colors), and classify them if all need be seen. Also, a user can open the attribute table to see all values, and also use the "Information" button of ArcMap to drill into data layers. (To check consistency of the fragmented data with the original MAP, compare the number of pixels with 0, from the "attribute table" menu (right-click on the layer in the Table of Contents).)

   Users are encouraged users to follow workaround (ii).

2) On occasions, after changing between CIDs, a MAP selection failed to display. The new layer is in the Table of Contents of ArcMap, but the layer showed the previous CID dimensions. Exiting ArcMap and

deleting the old MAP1.* from %CIDTEMP% solved the problem. The code has been amended, and the problem has not since been observed. (The change entailed moving all layer files to the database\metrics\layers folder.)

If ArcMap in future fails in the CID code, or in displaying a MAP with CID data then it is suggested that all files in the %CIDTEMP% be deleted, or more cautiously, %CIDTEMP% be redefined to an empty folder.